Universidad de las Ciencias Informáticas

Facultad # 15

Título:

Requisitos y Diseño del módulo de Configuración del proyecto SINAPSIS.

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

Autor: Minardo Fuerte Infante

Danis Diosdado Pérez Luis

Tutor: Ing. Daynier Ruiz Rodríguez

Co-Tutor: Ing. José Miguel Cazorla Santana

Ciudad de la Habana, Cuba

junio de 2010.

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la facultad # 15 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Minardo Fuerte Infante

_____

Autor.

Danis Diosdado Pérez Luis

_____

Autor.

Daynier Ruiz Rodríguez

_____

Tutor.

José Miguel Cazorla Santana

_____

Co-Tutor.

Agradecimientos

A esa persona que ya antes de yo venir a este mundo me estaba entregando parte de su vida, mi madre, por todo ese amor que me ha dado, es lo más puro que existe. Por su empeño, sacrificio y dedicación; gracias por existir y estar siempre a mi lado, te quiero mucho.

A mi papá por todo su apoyo, confianza y amor; por sentirse siempre orgulloso de mí, por ser ese gran amigo y el mejor padre del mundo, ha sido muy importante para mí saber que siempre estás ahí, te quiero mucho viejuco.

A mi abuela por quererme tanto y ser como una madre, Mamá Gina todo tu cariño y amor me han hecho una mejor persona, sé que tus pensamientos siempre me guían por el buen camino y que has dado todo de ti y más por ayudarme en la vida, por eso te quiero tanto.

A yumi por estar siempre conmigo y apoyarme en estos últimos años de mi carrera, no sé que hubiera sido de mí sin ti, los días que pasamos juntos nunca los olvidare, como tampoco a ti, gracias por tantas cosas, pero sobre todo por abrirme las puertas a ese sentimiento tan lindo que es enamorarse y por hacerme sentir tan especial.

A mi hermana por su apoyo y cariño, y por darme dos sobrinos tan lindos, y a ellos por alegrarme la vida, los quiero mucho.

A Manolo por acompañar a mi mamá en todo este tiempo que he estado lejos, y por tenerme como un hijo, muchas gracias por todo, me ha ayudado mucho.

A mi familia en general por todo su cariño, confianza y apoyo.

A mi tutor Daynier Ruiz Rodríguez, el co-tutor José Miguel Cazorla Santana y todos los profesores que ayudaron en la realización de este trabajo.

Danis

A mis padres, mi hermana, y a toda mi familia y amigos por influir en lo que soy.

A la Revolución y a Fidel por haber creado esta Universidad siendo esta la que me ha formado como profesional.

A mi tutor y co-tutor quienes conjuntamente con los profesores que me formaron han contribuido en este trabajo.

A todas aquellas personas que tanto dentro como fuera de la Universidad me han apoyado o ayudado de alguna forma.

A todos ustedes muchísimas gracias.

minar2

Dedicatoria

A mis padres, porque a ellos les debo todo lo que soy.

Danis

A mis padres por haberme apoyado en todo momento.

A mi hermana.

A toda mi familia y amigos.

A todos los que alguna vez me ayudaron a llegar donde estoy.

A ustedes está dedicado este trabajo.

minar2

Resumen

En la República Bolivariana de Venezuela se registran a través del sistema Nueva Etapa todos los proyectos que son formulados y planificados. Se puede afirmar que actualmente Nueva Etapa no satisface todas las expectativas de los clientes y usuarios, dado que no existe un proceso homogéneo de gestión de proyectos para todos los órganos y entes centralizados y descentralizados.

Para darle solución a esta problemática se decidió desarrollar el Sistema Nacional Público para el Seguimiento de Inversiones y Sectores. Un sistema informático que sustituirá completamente el anterior, para el desarrollo de este proyecto se ha dividido el trabajo en 7 módulos, entre los que se encuentra el de Configuración. El módulo Configuración permitirá a los clientes y usuarios configurar el sistema dependiendo de las cambiantes formas en que se realizan los proyectos.

Para lograr un eficiente desarrollo del módulo se realizó un estudio del estado del arte seleccionando como metodología de desarrollo RUP (Rational Unified Process), como lenguaje de modelado UML (Unified Modeling Language), como lenguaje de programación Java, y como herramienta CASE (Computer Aided Software Engineering) Visual Paradigm for UML.

Tras haber seleccionado la metodología, lenguaje de modelado, lenguaje de programación y herramienta CASE a utilizar se realizaron la especificación de requisitos, el modelo del sistema y el modelo del diseño para facilitar el trabajo de los desarrolladores en la implementación. Después de construidos los artefactos se validaron los mismos con el objetivo de comprobar la calidad de estos.

Índice de figuras

Índice

Introducción

La República Bolivariana de Venezuela desde el primero de enero de 2006 lleva a cabo el presupuesto por proyectos a solicitud de su Presidente, el Comandante Hugo Rafael Chávez Frías, en julio del año 2005. A través del presupuesto el 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 (MPPPD), conjuntamente con el Ministerio de Economía y Finanzas y la Vicepresidencia de la República, decidieron implantar un sistema informático que facilitará el registro, seguimiento y control de todos los proyectos que formarían parte del presupuesto anual de la nación, denominado Nueva Etapa.

A través del sistema antes mencionado se registran todos los proyectos que son formulados y planificados por los entes, tanto centralizados como descentralizados. La suma del presupuesto de un año de los proyectos aprobados por la máxima dirección de la administración pública es lo que se conoce como presupuesto anual.

El sistema Nueva Etapa no fue desarrollado utilizando una metodología de desarrollo de software, y no cuenta con la documentación necesaria que facilite la comprensión y optimización de dicho sistema. A pesar de ser una aplicación web no está disponible para sus usuarios durante todo el año, por lo que se formulan y planifican los proyectos con estimaciones de costos y tiempos irreales.

Se puede afirmar que actualmente Nueva Etapa no satisface todas las expectativas de los clientes y usuarios, dado que no existe un proceso homogéneo de gestión de proyectos para todos los órganos y entes centralizados y descentralizados. Además, no están bien definidos los flujos y niveles de aprobación por los que transita un proyecto. Venezuela es un país revolucionario, en constante transformación, debido a esto, las necesidades de la administración pública son cada vez más exigentes.

Para darle solución a esta problemática se decidió desarrollar el Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS). Un sistema informático que sustituirá completamente el anterior, para el desarrollo de este proyecto se ha dividido el trabajo en 7 módulos, entre los que se encuentra el de Configuración.

En el módulo antes mencionado para minimizar la complejidad del uso del sistema y así garantizar una mayor efectividad del mismo es necesario permitir a los clientes y usuarios configurar el sistema

dependiendo de las cambiantes formas en que se realizan los proyectos por lo que es indispensable una eficiente captura de los requisitos de software para así lograr una mayor satisfacción del cliente.

El correcto manejo de los requisistos identificados permitirá en gran medida la realización de un buen diseño del sistema, así como la comprensión tanto de los clientes y usuarios como de los desarrolladores de las distintas funcionalidades que sean identificadas en el módulo Configuración, además todo esto facilitará el trabajo en fases posteriores como la implementación.

Problema científico

Los documentos actuales del módulo Configuración 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.

Objetivo

Generar los documentos necesarios del módulo Configuración del sistema SINAPSIS que permitan el comienzo y continuidad de la implementación del módulo.

Tareas investigativas

1. Elaborar marco teórico para la justificación del trabajo.
2. Analizar los requisitos de software con el fin de clasificarlos y eliminar todas las inconsistencias y falencias que los mismos puedan tener.
3. Construir el Diagrama de Casos de Uso del Sistema a partir de los requisitos obtenidos, así como la realización de cada uno de ellos.
4. Confeccionar los Diagramas de Clases del Diseño.
5. Confeccionar los Diagramas de Secuencia.
6. Confeccionar el Diagrama Entidad Relación.
7. Validar los resultados obtenidos para evaluar la calidad de los mismos.

Métodos y técnicas de investigación a utilizar

Métodos teóricos:

Histórico - Lógico: Para el análisis de la trayectoria de la los requisitos y el diseño y para expresar teóricamente sus elementos fundamentales.
Modelación: Para la creación de modelos como abstracciones, con el objetivo de explicar cómo funciona el proceso de configuración.

Métodos empíricos:
Entrevista: Para la obtención de información sobre los procesos de configuración.
Encuesta: Para la medición de la satisfacción del cliente en función de los requisitos del sistema.

Estructura de la Tesis

El presente trabajo está compuesto por tres capítulos, los cuales están estructurados de la siguiente manera:

Capítulo 1: En el desarrollo de este capítulo se hace un estudio del proceso de desarrollo de software por lo que se analizan las metodologías y tecnologías candidatas a ser utilizadas para dar solución al problema, seleccionando las más apropiadas. Además se analizan las opciones existentes de patrones de diseño, de caso de uso y patrones GRASP que podrían utilizarse en el trabajo para lograr un producto con mayor calidad.

Capítulo 2: Este capítulo contiene lo referente a los requisitos y el diseño del sistema. Como propuesta de solución se obtienen los siguientes artefactos: los actores y casos de uso, los diagramas de casos de uso del sistema así como la descripción de los casos de uso, la realización de los diagramas de clases; subsistemas y paquetes del diseño, diagramas de secuencia y el diagrama entidad relación.

Capítulo 3: Este capítulo contiene un análisis de los resultados mediante la aplicación de métricas para medir la calidad de cada uno de los principales artefactos obtenidos.

Capítulo 1 Fundamentación Teórica

1.1 Introducción

En el desarrollo de este capítulo se hace un estudio del proceso de desarrollo de software por lo que se analizan las metodologías y tecnologías candidatas a ser utilizadas para dar solución al problema, seleccionando las más apropiadas. Además se analizan las opciones existentes de patrones de diseño, de caso de uso y patrones GRASP que podrían utilizarse en el trabajo para lograr un producto con mayor calidad.

1.2 Ingeniería de Software

A finales de los años 60, con el vertiginoso aumento en el desarrollo del hardware, el mismo deja de ser un impedimento para el desarrollo de la informática, reduciendo así los costos y mejorando la calidad y eficiencia del software producido, todo esto trae como resultado una crisis en el desarrollo de software la cual traía aparejado los siguientes problemas:

- Imprecisión en la planificación de los proyectos así como en la estimación de los costos de los mismos.
- Baja calidad del software.
- Dificultad para el mantenimiento de programas con un diseño poco estructurado. (1)

Para dar solución a todos estos problemas un grupo de estudio de la OTAN (Organización del Tratado del Atlántico Norte) acuña el término “Ingeniería de Software” teniendo como objetivo:

- Mejorar la calidad de los productos de software.
- Aumentar la productividad y trabajo de los ingenieros de software.
- Facilitar el control del proceso de desarrollo de software.
- Suministrar a los desarrolladores las bases para construir software de alta calidad en una forma eficiente.
- Definir una disciplina que garantice la producción y el mantenimiento de los productos software desarrollados en el plazo fijado y dentro del costo estimado. (2)

En la actualidad se pueden encontrar varios conceptos de Ingeniería de Software, a continuación se muestran algunos de ellos:

- Según Fritz Bauer en la primera conferencia sobre desarrollo de software patrocinada por el Comité de Ciencia de la OTAN celebrada en Garmisch, Alemania, en octubre de 1968. (3)

  “La definición, y el uso de principios y métodos de Ingeniería para obtener software económico, confiable y que trabaje sobre máquinas reales”

- Según Bohem, 1976. (4)
  "La aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos"

- Según Zelkovitz, 1978. (5)
  "El estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software."

- Según la IEEE (The Institute of Electrical and Electronics Engineers) una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías 1993. (6)
  "La aplicación de un enfoque cuantificable, disciplinado y sistemático al desarrollo, operación y mantenimiento del software."

- Según Pressman. (7)
  "Una tecnología multicapa en la que se pueden identificar: los métodos (indican cómo construir técnicamente el software), el proceso (es el fundamento de la Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos)"

Fig. 1 La Ingeniería de Software es una Tecnología Multicapa.

Después de haber realizado este estudio se puede llegar a la conclusión de que la Ingeniería de Software facilita el desarrollo de un proyecto de software, evita los atrasos ya que se puede planificar mejor el trabajo y por lo tanto disminuir los costos, aumentar la calidad del producto final, así como permitir la actualización y mantenimiento del software.

1.3 Ingeniería de Requisitos

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. El proceso de ingeniería de requisitos puede ser descrito en cinco pasos distintos:

- Identificación de Requisitos.
- Análisis de Requisitos y Negociación.
- Especificación de Requisitos.
- Modelado del Sistema.
- Validación y Gestión de Requisitos. (7)

Algunas definiciones de Ingeniería de Requisitos:

- Es un atributo necesario en un sistema, una declaración que identifica la capacidad, características, o factor de calidad de un sistema a fin de que tenga valor y utilidad a un cliente o usuario. (8)
- Una condición o capacidad necesaria dada por un usuario con el objetivo de resolver un problema o alcanzar un objetivo.
- Expresan las necesidades y restricciones atribuibles a un producto de software que contribuye a la solución de algún problema del mundo real. (9)

1.3.1 Técnicas Empleadas en la Captura de Requisitos

La Captura de Requisitos es una actividad algo compleja porque necesita la extracción de una serie de datos en entrevistas con los clientes y muchas veces los clientes no están seguros de lo que desean. Por eso es que se ha trabajado para tratar de establecer técnicas que permitan hacer el proceso de forma eficiente y eficaz.

Esta actividad es la pieza fundamental del proyecto marcando el punto de partida para las siguientes actividades del mismo, sirviendo de base para verificar si se alcanzaron los objetivos establecidos en el proyecto.

- **Entrevistas.** Las entrevistas son la técnica de elicitación más utilizada, y de hecho son prácticamente inevitables en cualquier desarrollo ya que son una de las formas de comunicación más naturales entre personas. (10) La entrevista no es una técnica sencilla de aplicar, requiere que el entrevistador sea experimentado y tenga capacidad para elegir bien a los entrevistados y obtener de ellos toda la información posible en un período de tiempo siempre limitado. Es por ello que la preparación de la entrevista juega un papel fundamental. (11)

En el libro de Pressman se presentan conjuntos de preguntas que se pueden utilizar en el desarrollo de esta técnica, que tiene una alta participación del analista y se realiza en conjunto con otras técnicas. (7)

- **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. Puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de captura, cuando los requisitos son todavía muy difusos.

  Consiste en la mera acumulación de ideas y/o información sin evaluar las mismas. Como técnica de captura de requisitos es sencilla de usar y de aplicar. Además suele ofrecer una visión general de las necesidades del sistema, pero normalmente no sirve para obtener detalles concretos del mismo, por lo que suele aplicarse en los primeros encuentros. (11)

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

  La ventaja esencial de los casos de uso es que resultan muy fáciles de entender para el usuario o cliente, sin embargo, carecen de la precisión necesaria si no se acompañan con una información textual o detallada con otra técnica, como pueden ser los diagramas de actividades. (11)

- **Cuestionarios.** Esta técnica requiere que el analista conozca el ámbito del problema en el que está trabajando. Consiste en redactar un documento con preguntas cuyas respuestas sean cortas y concretas, o incluso cerradas por unas cuantas opciones en el propio cuestionario (Checklist). Este cuestionario será cumplimentado por el grupo de personas entrevistadas o simplemente para recoger información en forma independiente de una entrevista. (11)

- **Comparación de terminología:** Uno de los problemas que surge durante la elicitación de requisitos es que usuarios y expertos no llegan a entenderse debido a problemas de terminología. Esta técnica es utilizada en forma complementaria a otras técnicas para obtener consenso respecto de la terminología a ser usada en el proyecto de desarrollo. Para ello es necesario identificar el uso de términos diferentes para los mismos conceptos (correspondencia), misma terminología para diferentes

conceptos (conflictos) o cuando no hay concordancia exacta ni en el vocabulario ni en los conceptos (contraste). (11)

1.4 Proceso de Desarrollo de Software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que cumpla con los requisitos del cliente. Dicho proceso, en términos globales se muestra en la Fig. 2. El proceso permite organizar el trabajo del equipo de desarrollo, ya que define las tareas a realizar por cada integrante para construir las piezas o artefactos necesarios, así como la forma y el momento en se van a realizar. (12)

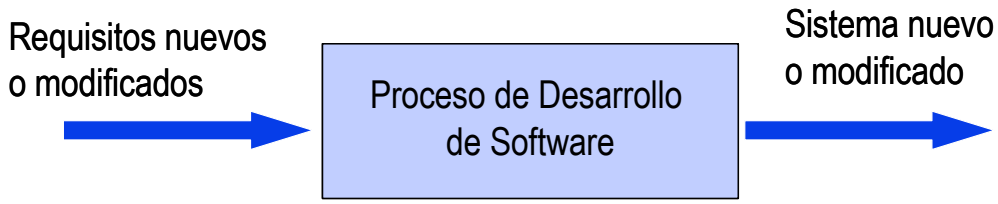


Fig. 2 Proceso de Desarrollo de Software.

1.4.1 Tendencias y Metodologías del Desarrollo de Software

Hoy en día el mercado está mucho más informado sobre los productos que necesita y reclama que:

- **El software debe ser efectivo**, debe hacer algo útil para el cliente. De nada sirve todo un año de desarrollo (¡o más!) para terminar entregando algo que al cliente no le sirve, ya sea por resultar obsoleto o porque no es lo que se imaginaba.

- **El software debe ser confiable**, debe funcionar bien. El software que se entrega es la cara visible del esfuerzo del equipo, y una herramienta de trabajo para el cliente. Parte de su negocio depende de este software, por lo que se debe entregar un producto que honre la relación que se establecen con él.

- **El software debe entregarse a tiempo**, en los plazos a los que se acordaron. El cliente planifica y coordina sus acciones en base a los compromisos y promesas asumidas por sus socios. (13)

En este sentido la Industria del Software en los últimos años ha evolucionado en lo que respecta a las metodologías de desarrollo de software, aumentando la tendencia a que el cliente tenga más participación y también la reducción de algunos procedimientos innecesarios que saturaban el proceso de desarrollo.

En 1998 Pressman planteó una serie de problemas que afectan sustancialmente el buen desarrollo de software, estos se citan a continuación:

- No se utilizan eficazmente las metodologías modernas de desarrollo del software ni las herramientas de Ingeniería del Software asistida por computadora (CASE), que son más importantes que el hardware más moderno para conseguir una buena calidad y productividad.

- No existe una adecuada descripción formal y detallada del ámbito de la información, funciones, rendimiento, interfaces, ligaduras de diseño y criterios de validación por una mala comunicación entre clientes y analistas.

- No se recogen datos sobre el proceso de desarrollo del software.

- No se garantiza la calidad desde el inicio del proyecto, ni se aplica la revisión técnica formal que es un filtro de calidad muy efectivo para encontrar defectos en el software. (7)

Actualmente existen un gran número de metodologías las cuales se agrupan en dos categorías: las metodologías Tradicionales y las Ágiles**.** Se realizará un estudio de las principales que se usan en el desarrollo de software con el objetivo de seleccionar la más adecuada para guiar el proceso de desarrollo del presente trabajo.

1.4.1.1 Metodologías Ágiles

Las metodologías ágiles están orientadas a proyectos pequeños, estas buscan reducir el tiempo de desarrollo pero sin descuidar la calidad del producto. Los propulsores de este enfoque firmaron un documento donde se expresan los principios fundamentales de esta metodología:

- Valorar a las personas y su interacción, por encima de los procesos y las herramientas, procesos de calidad con personas y relaciones mediocres no darán buenos resultados.

- Valorar el software que funciona, por encima de la documentación exhaustiva, la documentación es necesaria dado que permite la transferencia del conocimiento, pero su redacción debe limitarse a aquello que aporte valor directo al producto/servicio.

- Valorar la colaboración con el cliente, por encima de la negociación contractual: si bien son necesarios, los contratos no aportan valor a los productos/servicios. Las metodologías ágiles integran al cliente en el proyecto y mantienen como objetivo aportar el mayor valor posible en cada iteración.

- Valorar la respuesta al cambio, por encima del seguimiento de un plan: anticipación y adaptación enfrente de planificación y control. (14)

Ejemplos de metodologías ágiles:

- **Extreme Programing (XP).** La metodología ágil más radical y popular, se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promueve el trabajo en equipo y se preocupa por el aprendizaje de los desarrolladores.

- **SCRUM:** Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos.

- **Dynamic Systems Development Method (DSDM):** Define el marco para desarrollar un proceso de producción de software.

- **Adaptive Software Development (ASD):** Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. (14)

1.4.1.2 Metodologías Robustas

Las metodologías tradicionales o robustas son más rígidas, se centran en el control estricto de todo el proceso, estableciendo las actividades a realizar, los artefactos que se deben producir, las herramientas y un interés especial en llevar una documentación exhaustiva de todo el proyecto. Estas metodologías han demostrado su efectividad, principalmente en proyectos grandes, pero también presentan algunos problemas, como son los altos costos a la hora de implementar un cambio y la falta de flexibilidad en proyectos donde el entorno es volátil.

Ejemplos de metodologías robustas:

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

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

- **Microsoft Solution Framework (MSF):** metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. (15)

1.4.2 Extreme Programing (XP)

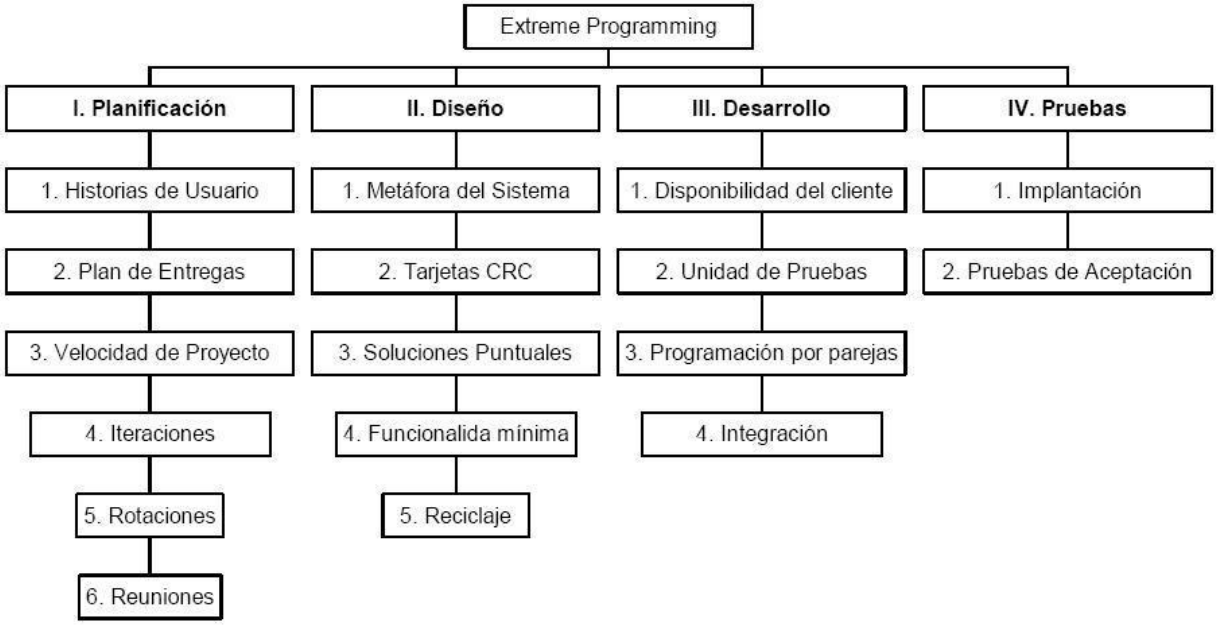


Fig. 3 Fases de la Metodología XP.

La metodología ágil más radical y popular, se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promueve el trabajo en equipo y se preocupa por el aprendizaje de los desarrolladores. También promueve la comunicación continua entre el cliente y el equipo de desarrollo, y es muy flexible para implementar cambios. Es adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Entre sus características más importantes están la simplicidad, la comunicación y la reutilización del código desarrollado.

XP surgió como respuesta y posible solución a los problemas derivados del cambio en los requisitos, esta se plantea como una metodología a emplear en proyectos de riesgo y favorece que aumente la productividad. (16)

“Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar”. (13)

1.4.3 Proceso Unificado de Desarrollo (RUP)

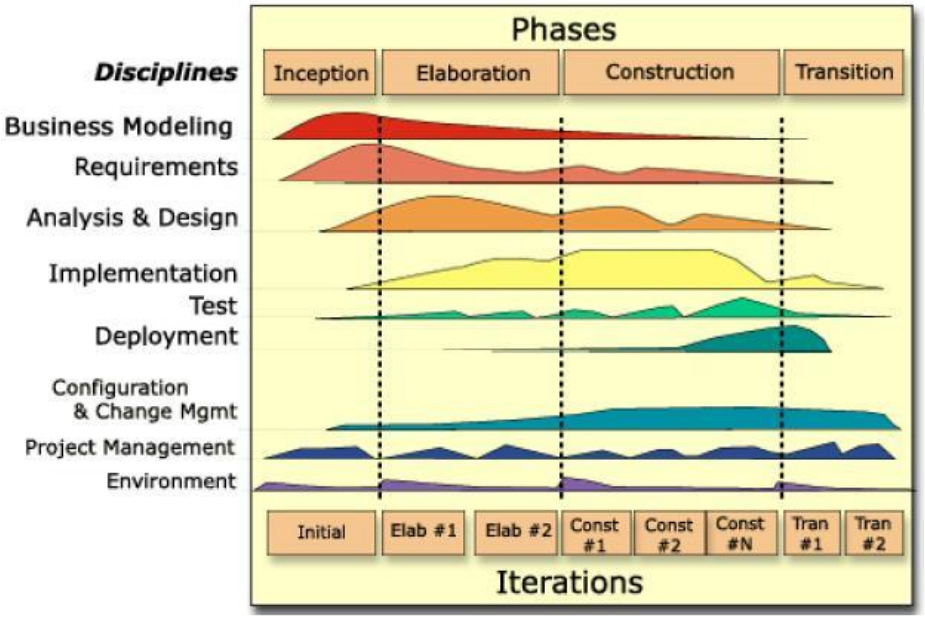


Fig. 4 El Ciclo de Vida de RUP.

RUP es un proceso de software genérico que puede ser utilizado para una gran cantidad de tipos de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de competencia y diferentes tamaños de proyectos.

La primera dimensión de la Fig. 4 representa el aspecto dinámico del proceso conforme se va desarrollando, se expresa en términos de fases, iteraciones e hitos.

Las Cuatro Faces son:

- **Conceptualización (Concepción o Inicio):** Tiene por finalidad definir la visión, los objetivos y el alcance del proyecto, tanto desde el punto de vista funcional como del técnico, obteniéndose como uno de los principales resultados una lista de los Casos de Uso y una lista de los factores de riesgo del proyecto. El principal esfuerzo está radicado en el “Modelamiento del Negocio” y el “Análisis de Requisitos”.

- **Elaboración:** Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requisitos (funcionales y no funcionales) identificados de acuerdo al alcance definido.

- **Construcción:** Está compuesta por un ciclo de varias iteraciones, en las cuales se van incorporando sucesivamente los casos de uso, de acuerdo a los factores de riesgo del proyecto. Este enfoque permite por ejemplo contar en forma temprana con versiones el sistema que satisfacen los principales Casos de Uso. Los cambios en los requisitos no se incorporan hasta el inicio de la próxima iteración.

- **Transición:** El release ya está listo para su instalación en las condiciones reales. Puede implicar reparación de errores al sistema que el usuario se encuentra utilizando activamente. (12)

La segunda dimensión de la Fig. 4 representa el aspecto estático del proceso: cómo es descrito en términos de componentes del proceso, disciplinas, actividades, flujos de trabajo, artefactos y roles.

Los Flujos de Trabajo son:

- **Modelamiento del Negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.

- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

- **Análisis y Diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requisitos), por lo que indica con precisión lo que se debe programar.

- **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.

- **Prueba (Testeo):** Busca los defectos a los largo del ciclo de vida.

- **Instalación o Despliegue:** Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

- **Administración del Proyecto:** Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

- **Administración de Configuración y Cambios:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

- **Ambiente:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

RUP usa el Lenguaje de Modelado Unificado (UML) en la preparación de todos los planos del sistema. De hecho, UML es una parte integral de RUP, fueron desarrollados a la par. Los aspectos distintivos de RUP están capturados en tres conceptos clave:

- **Dirigido por Casos de Uso:** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requisitos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

- **Centrado en la Arquitectura:** La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

- **Iterativo e Incremental:** Aunque la Fig. 4 puede sugerir que los flujos de trabajo se desarrollan en cascada, la "lectura" de este gráfico tiene que ser vertical y horizontal. RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. (12)

RUP provee a cada miembro del equipo de las guías de proceso, plantillas y mentores de herramientas necesarios para que el equipo completo tome ventaja de, entre otras, las siguientes mejores prácticas:

Fig. 5 Mejores Prácticas para el Desarrollo de Software.

- Desarrollar Software Iterativamente.
- Administrar los Requisitos.
- Utilizar Arquitecturas Basadas en Componentes.
- Modelar Software Visualmente.
- Verificar la Calidad de Software.
- Controlar los Cambios al Software. (17)

Tras haber realizado un estudio de las metodologías candidatas se decidió seleccionar la metodología RUP, entre otras cosas porque proporciona una infraestructura flexible de desarrollo de software que implementa algunas de las mejores prácticas recomendadas en la industria. Es un proceso iterativo e incremental lo que ayuda a mitigar los riesgos en forma temprana y continua, es dirigido por casos de usos, estos no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo y además es centrado en la arquitectura lo que permite tener una perspectiva clara del sistema completo desde los inicios ,que va a ser necesaria para controlar el desarrollo posterior.

Además como el objetivo final de esta tesis es realizar el diseño del módulo Configuración del proyecto SINAPSIS, se decidió utilizar RUP en lugar de la otra alternativa XP, pues este último es un proceso más orientado a la implementación, por lo que para lograr un diseño exitoso no sería la más adecuada, y otro motivo es que los clientes están demasiado lejos y XP propone una continua interacción con los clientes, RUP genera una gran cantidad de artefactos que facilitarán la futura implementación. Por otro lado el equipo tiene excelentes experiencias en trabajos anteriores utilizando la metodología RUP.

1.5 Lenguaje Unificado de Modelado (UML)

Durante los ochenta y principios de los noventa Grady Booch, James Rumbaugh, e Ivar Jacobson trabajaban por separado en desarrollo de notaciones para el análisis y diseño de sistemas orientados a objetos. Los tres llegaron por separado a obtener bastante reconocimiento.

Como objetivos principales de la consecución de un nuevo método que aunara los mejores aspectos de sus predecesores, sus protagonistas se propusieron lo siguiente:

- El método debía ser capaz de modelar no sólo sistemas de software sino otro tipo de sistemas reales de la empresa, siempre utilizando los conceptos de la orientación a objetos (OO).
- Crear un lenguaje para modelado utilizable a la vez por máquinas y por personas.
- Establecer un acoplamiento explícito de los conceptos y los artefactos ejecutables.
- Manejar los problemas típicos de los sistemas complejos de misión crítica.

Nuevas características de UML.

Además de los conceptos extraídos de métodos anteriores, se han añadido otros nuevos que vienen a suplir carencias antiguas de la metodología de modelado. Estos nuevos conceptos son los siguientes:

- Definición de estereotipos: un estereotipo es una nueva clase de elemento de modelado que debe basarse en ciertas clases ya existentes en el metamodelo y constituye un mecanismo de extensión del modelo.

- Responsabilidades.
- Mecanismos de extensibilidad: estereotipos, valores etiquetados y restricciones.
- Tareas y procesos.
- Distribución y concurrencia (para modelar por ejemplo ActiveX/DCOM y CORBA).
- Patrones/Colaboraciones.
- Diagramas de actividad (para reingeniería de proceso de negocios)
- Clara separación de tipo, clase e instancia.
- Refinamiento (para manejar relaciones entre niveles de abstracción).
- Interfaces y componentes. (18)

En la actualidad podemos encontrar varias definiciones de UML:

- Un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software. (12)

- Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. (19)

Se decide utilizar UML ya que la metodología seleccionada para desarrollar el trabajo (RUP) está basada en UML y como se ha podido apreciar brinda una gran cantidad de facilidades para el desarrollo de un proyecto de software.

1.6 Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering*, Ingeniería del Software Asistida por Ordenador), fueron creadas para dar soporte automatizado al desarrollo de software. El énfasis en planificación, análisis y diseño promovido por una herramienta CASE tiene un fuerte impacto y recompensa en la mejora de la calidad del producto obtenido y en el aumento de productividad (disminución de tiempos, costes y esfuerzos) en las actividades de desarrollo y mantenimiento.

De acuerdo con Kendall y Kendall (1997) la ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su

objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Las herramientas CASE son usadas en casi todas las fases del ciclo de desarrollo de software y los beneficios que provee su uso son considerables, algunos de estos son:

- Hacer el trabajo de diseño de software más fácil y agradable.
- Verificar el uso de todos los elementos en el sistema diseñado.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en las bases de datos.
- Generar estructuras de código.
- Reducción del costo de producción de software. (20)

Sin lugar a dudas las herramientas CASE han venido a revolucionar la forma de producir software, debido a la gran plataforma que ofrecen a los desarrollares, que hoy en día no conciben su trabajo sin utilizar una de éstas, por una parte la metodología brinda el plan a seguir y por otra las herramientas CASE se encargan de la parte técnica, automatizando muchas de las actividades definidas por las metodologías. La selección de ambas es de una importancia vital para el éxito del producto.

1.6.1 Rational Rose Enterprise Edition

Es una herramienta CASE que permite especificar, analizar y diseñar sistemas utilizando UML.

Algunas de sus características son:

- Mantiene la consistencia de los modelos del sistema software.
- Chequeo de la sintaxis UML.
- Generación de documentación automáticamente.
- Generación de código a partir de los modelos.
- Ingeniería Inversa (crear modelo a partir código).
- Diseño centrado en casos de uso y enfocado al negocio.
- Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables.
- Se ejecuta solo sobre el sistema operativo Windows.
- Desarrollo Iterativo. (21)

Se integra con herramientas Integrated Development Environment (IDE) como:

- Borland JBuilder versiones 7.0 a 10.0.
- Sun Forte for Java Community y Enterprise Editions.

- Microsoft Visual Studio 6.
- Microsoft Visual Studio 2003.
- Microsoft Visual Studio 2005.
- Wind River Tornado.
- Green Hills MULTI. (22)

Es una herramienta de diseño orientada a objetos, que da soporte al modelado visual, es decir, que permite representar gráficamente el sistema, permitiendo hacer énfasis en los detalles más importantes, centrándose en los casos de uso y enfocándose hacia un software de mayor calidad, empleando un lenguaje estándar común que facilita la comunicación.

1.6.2 Visual Paradigm for UML

Visual Paradigm es una herramienta CASE que utiliza UML como lenguaje de modelado, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y llevar la documentación.

Ofrece muchas ventajas como son:

- Entorno de creación de diagramas para UML 2.1.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Permite hacer los prototipos de interfaz de usuario.
- Disponibilidad de múltiples versiones, para cada necesidad. (23)

Plataforma Java (Windows/Linux/Mac OS X):

- SDE para Eclipse
- SDE para NetBeans
- SDE para Sun ONE
- SDE para Oracle JDeveloper
- SDE para JBuilder
- SDE para IntelliJ IDEA
- SDE para WebLogic Workshop

Plataforma Windows:

- SDE para Microsoft Visual Studio.

Después de analizar estas alternativas se decide utilizar Visual Paradigm como herramienta CASE ya que es una herramienta muy potente y fácil de utilizar, suma a su favor que cuenta con abundantes manuales, y una de sus ventajas más importantes es que es multiplataforma, tiene versiones para distintos sistemas operativos y también puede integrarse a varios entornos de desarrollo, permite la construcción de los prototipos no funcionales y además de todo la Universidad cuenta con una licencia para su uso.

1.7 Patrones

Un patrón es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto.

El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones.

Un buen patrón debería:

- Solucionar un problema.
- Ser un concepto probado.
- Cumplir con que la solución no sea obvia.
- Describir participantes y relaciones entre ellos.
- Tener un componente humano alto: estética y utilidad. (24)

1.7.1 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. Dado un contexto y un problema a resolver, estas técnicas han mostrado ser la solución adoptada en la comunidad del desarrollo de software.

Los patrones de casos de uso son los siguientes:

- Reglas de negocio.
- Concordancia (Commonality).
- Componente jerárquico (Component hierarchy).
- Extensión concreta o Inclusión
- CRUD (Creating, Reading, Updating, Deleting).
- Caso de uso grande (Large Use case).
- Sistema de capas.

- Múltiples actores.
- Servicio opcional.
- Vistas ortogonales.
- Secuencia de casos de uso.

A Continuación se Explican Algunos de Ellos:

- **Reglas de Negocio.** Se basan en la extracción de información originada de las políticas, reglas y regulaciones del negocio de la descripción del flujo y describe la información como una colección de reglas del negocio referenciadas a partir de las descripciones de los casos de uso.
  - **Definición estática**. Este patrón es aplicado a todos los casos de uso modelando los servicios que son afectados por las reglas del negocio definidas en la organización. Sin embargo, este patrón no influye en la estructura del modelo de casos de uso.
  - **Modificación dinámica.** Este modelo del patrón contiene un caso de uso llamado Gestionar regla, que se encarga de crear, actualizar y eliminar las reglas del negocio (Fig. 10).

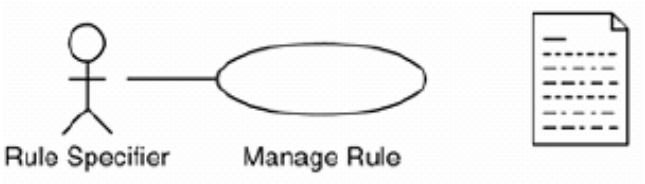


Fig. 6 Reglas del Negocio, Modificación Dinámica.

- **Concordancia (Commonality).** Extrae una subsecuencia de acciones que aparecen en diferentes lugares del flujo de casos de uso y es expresado por separado.
  - **Reusabilidad.** Consta de 3 casos de uso. El primero llamado subsecuencia común, modela una secuencia de acciones que aparecerán en múltiples casos de uso en el modelo.

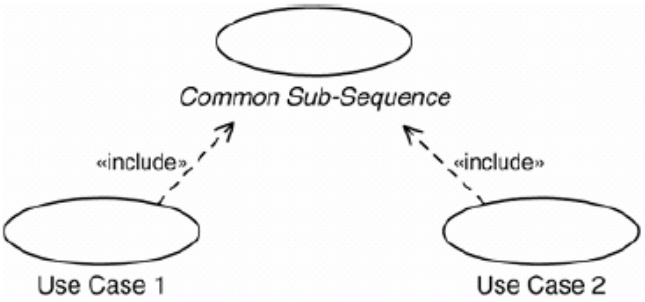


Fig. 7 Concordancia, Reusabilidad.

- **Adición.** En el caso de este patrón alternativo, la subsecuencia común de casos de uso, extiende los casos de uso compartiendo la subsecuencia de acciones. Los otros casos de uso modelan el flujo que será expandido con la subsecuencia.

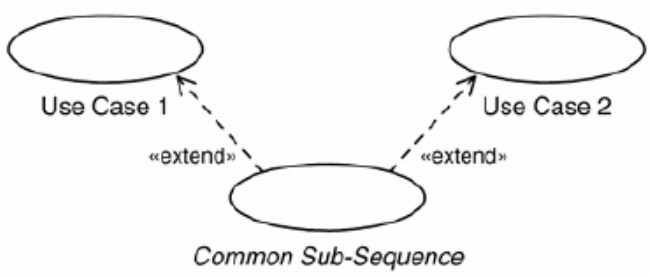


Fig. 8 Concordancia, Adición.

- **Especialización.** Otro patrón de concordancia que contiene casos de uso del mismo tipo. En este caso, estos son modelados como una especialización de casos de uso de tipo de uso común.

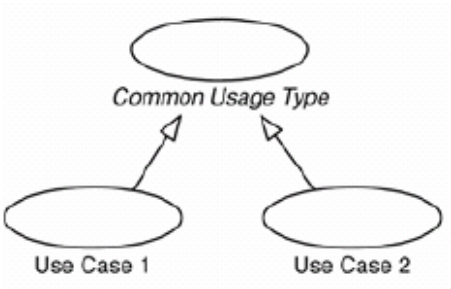


Fig. 9 Concordancia, Especialización.

- **Reusabilidad interna.** Si la subsecuencia de acciones es utilizada en diferentes lugares en un solo caso de uso, no existe la necesidad de extraer la subsecuencia dentro de un caso de uso separado.

- **CRUD (Creating, Reading, Updating, Deleting).** Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual.
  - **Completo.** Este patrón consta de un caso de uso, llamado Información CRUD o Gestionar información, modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación.

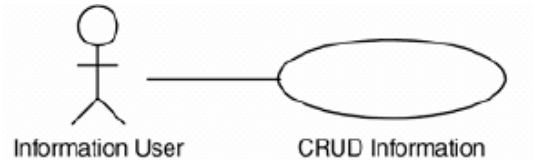


Fig. 10 CRUD, Completo.

- **Parcial.** Este patrón alternativo modela una de las vías de los casos de uso como un caso de uso separado.

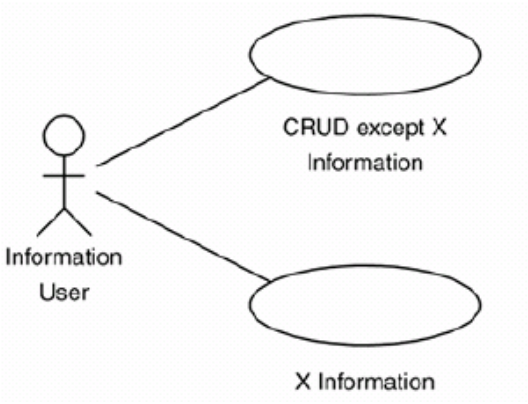


Fig. 11 CRUD, Parcial.

- **Múltiples Actores**
  - **Roles Diferentes.** Captura la concordancia entre actores manteniendo roles separados. Consiste de un caso de uso y por lo menos dos actores.

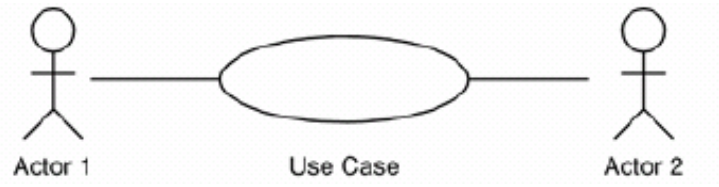


Fig. 12 Múltiples Actores, Roles Diferentes.

  - **Roles comunes.** Puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. (25)

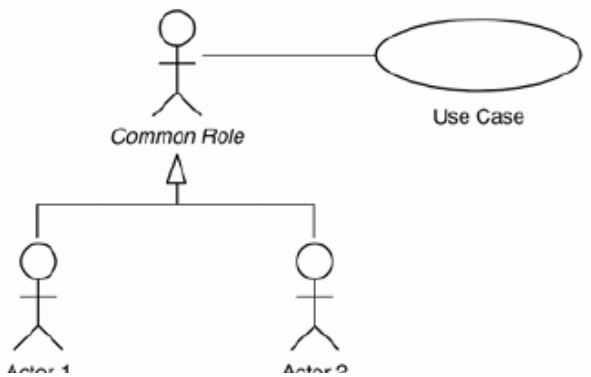


Fig. 13 Múltiples Actores, Roles Comunes.

1.7.2 Patrones de Diseño

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones. Surgieron en el año 1977, inventados por Christopher Alexander.

- **Patrones Creacionales.**
  - **Builder**: Abstrae el proceso de creación de un objeto complejo.
  - **Prototype**: Crea nuevos objetos clonándolos de una instancia ya existente.
  - **Singleton**: Garantiza la existencia de una única instancia para una clase.

- **Patrones Estructurales**
  - **Bridge**: Desacopla una abstracción de su implementación.
  - **Composite**: Permite tratar objetos compuestos como si de un simple se tratase.
  - **Decorator**: Añade funcionalidad a una clase dinámicamente.
  - **Facade**: Provee de una interfaz unificada simple para acceder a una interfaz.
  - **Flyweight**: Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
  - **Proxy**: Mantiene un representante de un objeto.

- **Patrones de Comportamiento**
  - **Command**: Encapsula una operación en un objeto.
  - **Interpreter**: Dado un lenguaje, define una gramática para dicho lenguaje.
  - **Mediator**: Define un objeto que coordine la comunicación entre objetos de distintas clases.
  - **Memento**: Permite volver a estados anteriores del sistema.
  - **Observer**: Define una dependencia de uno-a-muchos entre objetos.
  - **State**: Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
  - **Strategy**: Permite disponer de varios métodos para resolver un problema.

- **Template Method**: Define en una operación el esqueleto de un algoritmo. (26)

Definición de un patrón de diseño.

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

Ventajas:

- Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.
- Por tanto, están basados en la recopilación del conocimiento de los expertos en desarrollo de software.
- Es una experiencia real, probada y que funciona. Es Historia y nos ayuda a no cometer los mismos errores.

Características:

- Son soluciones concretas.
- Son soluciones técnicas.
- Se utilizan en situaciones frecuentes.
- Favorecen la reutilización de código.
- El uso de un patrón no se refleja en el código.
- Es difícil reutilizar la implementación de un patrón. (27)

1.7.2.1 Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

- **Patrón Experto:**

  Problema:

  ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?

  - Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades.

- Si estas se asignan en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y se nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones.

- **Patrón Creador:**

  Problema:

  ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

  - La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos.
  - En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella.
  - El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilización.
  - El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos.
  - El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.
  - Al escogerlo como creador, se da soporte al bajo acoplamiento.

- **Patrón Bajo Acoplamiento:**

  Problema:

  ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

  - El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas.
  - Acoplamiento bajo significa que una clase no depende de muchas clases.
  - Acoplamiento alto significa que una clase recurre a muchas otras clases.

- **Patrón Alta Cohesión:**

  Problema:

  ¿Cómo mantener la complejidad dentro de límites manejables?

  - La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase.
  - Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.
  - Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo.

- **Patrón Controlador:**

  Problema:

  ¿Quién debería encargarse de atender un evento del sistema?

  - Un evento del sistema es un evento de alto nivel generado por un actor externo; es un evento de entrada externa. Se asocia a operaciones del sistema: las que emite en respuesta a los eventos del sistema.

  - Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

  Un defecto frecuente al diseñar controladores consiste en asignarles demasiada responsabilidad. Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. (27)

1.8 Lenguajes de Programación Web

Actualmente existen diferentes lenguajes de programación para desarrollar en la web, estos han ido surgiendo debido a las tendencias y necesidades de las plataformas. Desde los inicios de Internet, fueron surgiendo diferentes demandas por los usuarios y se dieron soluciones mediante lenguajes estáticos. A medida que pasó el tiempo, las tecnologías fueron desarrollándose y surgieron nuevos problemas a dar solución. Esto dio lugar a desarrollar lenguajes de programación dinámicos para la web, que permitieran interactuar con los usuarios y utilizaran sistemas de Bases de Datos. A continuación se abordan algunos de los lenguajes de programación existentes para la web. (28)

PHP

PHP (Pre-procesador Hipertexto) es un lenguaje de programación implantado (enraizado) en HTML HTML-embebed scripting lenguaje. La mayoría de su sintaxis está prestada de los lenguajes de programación C, Java y Perl, con la inclusión de algunos rasgos únicos de PHP. La meta del lenguaje es permitir a los que desarrollan sitios Web escribir rápidamente páginas generadas dinámicamente. (29) Para su funcionamiento necesita tener instalado Apache o IIS con las librerías de PHP. Entre sus ventajas podemos encontrar que no requiere definición de tipos de variables ni manejo detallado del bajo nivel. y entre sus desventajas que la programación orientada a objetos es aún muy deficiente para aplicaciones grandes. (28)

JSP

Es una tecnología para la creación de sitios web dinámicos, acrónimo de Java Server Pages. Está orientado a desarrollar páginas web en Java. JSP es un tecnología multiplataforma. Creado para

ejecutarse del lado del servidor. JSP fue desarrollado por Sun Microsystems. Comparte ventajas similares a las de ASP.NET, desarrollado para la creación de aplicaciones web potentes. Posee un motor de páginas basado en los servlets de Java. Para su funcionamiento se necesita tener instalado un servidor Tomcat. Como desventaja podemos señalar la complejidad de aprendizaje y como ventajas que realiza la colección de basura de las ayudas, así que la gerencia de memoria es automática. (28)

Después de analizados ambos lenguajes se decide seleccionar a Java, por ser un lenguaje muy potente, los miembros del proyecto cuentan con experiencia desarrollando con dicho lenguaje y el equipo de arquitectura sugirió el uso del mismo.

1.9 Conclusiones Parciales

Actualmente los principales objetivos del proceso de desarrollo de software son: obtener productos con calidad, entregar a tiempo el producto, aumentar la productividad, poder controlar el proceso de desarrollo y llegado el momento dar mantenimiento al software producido. En la Universidad de las Ciencias Informáticas a pesar de la escasa experiencia en el desarrollo de software, continuamente se intenta mejorar el proceso, por lo que este capítulo se ha dedicado a hacer un estudio profundo del mismo, el cual abarca las metodologías de desarrollo, herramientas CASE, lenguajes de programación y lenguajes de modelado para hacer una correcta selección de cuáles de ellas guiarán el proceso de desarrollo del presente trabajo. Tras este estudio se decide seleccionar la metodología RUP, como lenguaje de modelado UML, como lenguaje de programación Java y como herramienta CASE Visual Paradigm for UML.

Capítulo 2 Requisitos y Diseño del Sistema

2.1 Introducción

Este capítulo contiene lo referente a los requisitos y el diseño del sistema. Como propuesta de solución se obtienen los siguientes artefactos: los actores y casos de uso, los diagramas de casos de uso del sistema así como la descripción de los casos de uso, la realización de los diagramas de clases; subsistemas y paquetes del diseño, diagramas de secuencia y el diagrama entidad relación.

Modelo del sistema

El modelo de sistema permite tener un entendimiento más detallado de cómo va a estar estructurado el sistema a partir de los CU identificados. Para ello se identifican los actores y se especifican los casos CUS. Este artefacto posibilita llegar a un acuerdo entre desarrolladores y clientes, además constituye una entrada fundamental para el análisis, diseño y pruebas.

Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos y restricciones relacionadas con el entorno de implementación tienen impacto en el sistema a considerar. Describe la jerarquía de subsistemas de diseño que contienen clases del diseño, realizaciones de casos de uso, diseño e interfaces. Sirve de abstracción de la implementación del sistema y es usado como una entrada fundamental de las actividades de implementación. (12)

2.2 Requisitos de Software

Durante esta etapa se registran los requisitos del software, y se especifican las capacidades operacionales que deberá tener el sistema lo más detalladamente posible. Los requisitos planteados persiguen como alcance llegar a un entendimiento entre el cliente y el equipo de desarrollo mostrando las condiciones que debe presentar el producto desde el punto de vista funcional. Los mismos se agruparon en dos categorías, funcionales y no funcionales. (31)

2.2.1 Requisitos Funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. A continuación se muestran los requisitos más importantes del módulo Configuración. Para acceder a la especificación íntegra de los requisitos, ver el documento Especificación de Requisitos del sistema SINAPSIS, Módulo Configuración. (32)

Algunas de las exigencias que el sistema debe satisfacer son:

RF_C.1 Crear entidad

El sistema permitirá crear una entidad a partir de los siguientes datos:

- Denominación
- Siglas
- Código
- Tipo de entidad
- Correo electrónico
- Teléfono
- Fax
- Dirección
- Logotipo
- Activo (tipo: booleano)

RF_C.2 Obtener entidad

El sistema permitirá obtener una entidad con los siguientes datos:

- Denominación
- Siglas
- Código
- Tipo de entidad
- Correo electrónico
- Teléfono
- Fax
- Dirección
- Logotipo
- Activo (tipo: booleano)

RF_C.3 Modificar entidad

El sistema permitirá modificar una entidad existente. Los campos a modificar serán los siguientes:

- Denominación
- Siglas
- Código
- Tipo de entidad
- Correo electrónico
- Teléfono
- Fax

- Dirección
- Logotipo
- Activo (tipo: booleano)

RF_C.4 Mostrar detalles de la entidad

El sistema permitirá mostrar los siguientes detalles de una entidad existente:

- Denominación
- Siglas
- Código
- Tipo de entidad
- Correo electrónico
- Teléfono
- Fax
- Dirección
- Logotipo
- Activo (tipo: booleano)

RF_C.5 Eliminar entidad

El sistema permitirá eliminar una entidad existente.

RF_C.6 Mostrar mensaje de confirmación

El sistema permitirá mostrar un mensaje de confirmación para que el usuario ratifique la realización de determinadas funcionalidades de impacto o definitorias del sistema.

RF_C.7 Mostrar mensaje informativo

El sistema permitirá mostrar un mensaje informativo para que el usuario conozca, por qué el sistema no le permite realizar la operación que desea.

RF_C.8 Mostrar mensaje flotante (tooltip)

El sistema permitirá mostrar un mensaje flotante (tooltip) para que el usuario conozca más sobre determinados campos de la interfaz que puedan resultarle ambiguos o de difícil comprensión.

2.2.2 Requisitos No Funcionales

Los requisitos no funcionales especifican propiedades o cualidades que el sistema debe tener. Estos representan las características del producto. A continuación se muestra una representación de los requisitos no funcionales del módulo Configuración. Para acceder a la especificación íntegra de los

requisitos, ver el documento Especificación de Requisitos No Funcionales del sistema SINAPSIS, Módulo Configuración. (33)

Usabilidad

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

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

Fiabilidad

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

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

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

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

Eficiencia

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

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

Seguridad

RNF.4 Restringir el acceso al sistema.

Para acceder al sistema cada usuario debe ser autenticado, de manera que se pueda determinar si tiene acceso al mismo y en caso de ser así determinar las funcionalidades a las que tiene acceso y restringir su actividad al uso de las mismas, las cuales están determinadas por los roles que contiene el usuario. Para autenticarse el usuario debe colocar: nombre de usuario y contraseña.

Portabilidad

RNF.5 El sistema debe ser una aplicación Web

El sistema se debe ejecutar sobre un entorno web de manera que se permita su acceso desde cualquier punto del país utilizando la red.

Reusabilidad

RNF.6 Garantizar que los formatos de los archivos de salida del sistema sean compatibles con los programas más comunes.

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

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

Capacidad

RNF.7 Considerar características técnicas mínimas para la ejecución en clientes

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

2.3 Actores del Sistema

Los actores son terceros fuera del sistema que interactúan directamente con él. (31) A continuación se describen los actores del sistema.

Fig. 14 Actores del Sistema.

Tabla. 1 Actores del Sistema

Actor	Descripción
Administrador central	El actor **"**Administrador central" es un usuario del sistema que además de poder hacer las mismas funciones que un Administrador de organismo, tiene permisos para gestionar (leer, crear, modificar y eliminar) aquellos objetos que pueden cambiar con el paso del tiempo, tales como: entidades, fuentes de financiamientos, contactos, sectores y otros que se mencionan a continuación de cada uno de los casos de usos descritos.

2.4 Casos de Uso del Sistema

Los casos de usos constituyen un conjunto de acciones que un sistema ejecuta y produce un resultado observable para un actor. (31) A continuación se enuncian cada uno de los casos de uso determinados.

Tabla. 2 CU: Gestionar tipo de entidad

Caso de Uso:	Gestionar tipo de entidad
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un tipo de entidad. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo tipo de entidad y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un tipo de entidad ya sea para modificarla, ver sus detalles o eliminarla. El caso de uso termina con la creación, modificación o eliminación de un tipo de entidad.
Referencias	RF_C.1, RF_C.2, RF_C.3, RF_C.4, RF_C.5, RF_C.78, RF_C.79, RF_C.80

Tabla. 3 CU: Gestionar entidad

Caso de Uso:	Gestionar entidad
Actores:	Administrador central

Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar una entidad. Consiste en que el Administrador central selecciona la opción de adicionar una nueva entidad y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar una entidad ya sea para modificarla, ver sus detalles o eliminarla. El caso de uso termina con la creación, modificación o eliminación de una entidad.
Referencias	RF_C.6, RF_C.7, RF_C.8, RF_C.9, RF_C.10, RF_C.78, RF_C.79, RF_C.80

Tabla. 4 CU: Gestionar fuente de financiamiento

Caso de Uso:	Gestionar fuente de financiamiento
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central de una entidad necesita gestionar una fuente de financiamiento. Consiste en que el Administrador central selecciona la opción de adicionar una nueva fuente de financiamiento y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar una fuente de financiamiento ya existente ya sea para modificarla o eliminarla. El caso de uso termina con la creación, modificación o eliminación de una fuente de financiamiento.
Referencias	RF_C.11, RF_C.12, RF_C.13, RF_C.14, RF_C.15, RF_C.78, RF_C.79, RF_C.80

Tabla. 5 CU: Gestionar asunto del comentario

Caso de Uso:	Gestionar asunto del comentario
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un asunto del comentario. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo asunto del comentario y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un asunto del comentario ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un asunto del comentario.
Referencias	RF_C.16, RF_C.17, RF_C.18, RF_C.19, RF_C.20, RF_C.78, RF_C.79, RF_C.80

Tabla. 6 CU: Gestionar contacto

Caso de Uso:	Gestionar contacto

Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un contacto (contáctenos). Consiste en que el Administrador central selecciona la opción de adicionar un nuevo contacto y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un contacto ya sea para modificarlo o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un contacto.
Referencias	RF_C.21, RF_C.22, RF_C.23, RF_C.24, RF_C.25, RF_C.78, RF_C.79, RF_C.80

Tabla. 7 CU: Gestionar tipo de proyecto

Caso de Uso:	Gestionar tipo de proyecto
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un tipo de proyecto. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo tipo de proyecto y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un tipo de proyecto ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un tipo de proyecto.
Referencias	RF_C.26, RF_C.27, RF_C.28, RF_C.29, RF_C.30, RF_C.78, RF_C.79, RF_C.80

Tabla. 8 CU: Gestionar tipo de inversión

Caso de Uso:	Gestionar tipo de inversión
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un tipo de inversión. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo tipo de inversión y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un tipo de inversión ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un tipo de inversión.
Referencias	RF_C.31, RF_C.32, RF_C.33, RF_C.34, RF_C.35, RF_C.78, RF_C.79, RF_C.80

Tabla. 9 CU: Gestionar lineamiento

Caso de Uso:	Gestionar lineamiento

Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un lineamiento. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo lineamiento y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un lineamiento ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un lineamiento.
Referencias	RF_C.36, RF_C.37, RF_C.38, RF_C.39, RF_C.40, RF_C.78, RF_C.79, RF_C.80

Tabla. 10 CU: Gestionar sector

Caso de Uso:	Gestionar sector
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un sector. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo sector y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un sector ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un sector.
Referencias	RF_C.41, RF_C.42, RF_C.43, RF_C.44, RF_C.45, RF_C.78, RF_C.79, RF_C.80

Tabla. 11 CU: Gestionar sub-sector

Caso de Uso:	Gestionar sub-sector
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un sub-sector. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo sub-sector y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un sub-sector ya sea para modificarlo, ver sus detalles o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un sub-sector.
Referencias	RF_C.46, RF_C.47, RF_C.48, RF_C.49, RF_C.50, RF_C.78, RF_C.79, RF_C.80

Tabla. 12 CU: Gestionar ámbito de localización estado

Caso de Uso:	Gestionar ámbito de localización estado
Actores:	Administrador central

Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un ámbito de localización estado. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo ámbito de localización estado y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un ámbito de localización estado ya sea para modificarlo o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un ámbito de localización estado.
Referencias	RF_C.51, RF_C.52, RF_C.53, RF_C.54, RF_C.78, RF_C.79, RF_C.80

Tabla. 13 CU: Gestionar ámbito de localización municipio

Caso de Uso:	Gestionar ámbito de localización municipio
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un ámbito de localización municipio. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo ámbito de localización municipio y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un ámbito de localización municipio ya sea para modificarlo o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un ámbito de localización municipio.
Referencias	RF_C.55, RF_C.56, RF_C.57, RF_C.58, RF_C.78, RF_C.79, RF_C.80

Tabla. 14 CU: Gestionar ámbito de localización parroquia

Caso de Uso:	Gestionar ámbito de localización parroquia
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un ámbito de localización parroquia. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo ámbito de localización parroquia y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un ámbito de localización parroquia ya sea para modificarlo o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un ámbito de localización parroquia.
Referencias	RF_C.59, RF_C.60, RF_C.61, RF_C.62, RF_C.78, RF_C.79, RF_C.80

Tabla. 15 CU: Gestionar ámbito de localización centro poblado

Caso de Uso:	Gestionar ámbito de localización centro poblado
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar un ámbito

	de localización centro poblado. Consiste en que el Administrador central selecciona la opción de adicionar un nuevo ámbito de localización centro poblado y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar un ámbito de localización centro poblado ya sea para modificarlo o eliminarlo. El caso de uso termina con la creación, modificación o eliminación de un ámbito de localización centro poblado.
Referencias	RF_C.63, RF_C.64, RF_C.65, RF_C.66, RF_C.78, RF_C.79, RF_C.80

Tabla. 16 CU: Gestionar zona económica de desarrollo sustentable

Caso de Uso:	Gestionar zona económica de desarrollo sustentable
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar una zona económica de desarrollo sustentable. Consiste en que el Administrador central selecciona la opción de adicionar una nueva zona económica de desarrollo sustentable y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar una zona económica de desarrollo sustentable ya sea para ver sus detalles, modificarla, ver sus detalles o eliminarla. El caso de uso termina con la creación, modificación o eliminación de una zona económica de desarrollo sustentable.
Referencias	RF_C.67, RF_C.68, RF_C.69, RF_C.70, RF_C.71, RF_C.78, RF_C.79, RF_C.80

Tabla. 17 CU: Gestionar partida presupuestaria

Caso de Uso:	Gestionar partida presupuestaria
Actores:	Administrador ONAPRE
Resumen:	El caso de uso se inicia cuando el Administrador ONAPRE necesita gestionar una partida presupuestaria. Consiste en que el Administrador ONAPRE selecciona la opción de adicionar una partida presupuestaria y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar una partida presupuestaria ya sea para modificarla, eliminarla, ver sus detalles o habilitarla para aquellos organismos que no la visualizan y así se desee. El caso de uso termina con la creación, actualización o eliminación de una partida presupuestaria.
Referencias	RF_C.72, RF_C.73, RF_C.74, RF_C.75, RF_C.76, RF_C.77, RF_C.78, RF_C.79, RF_C.80

2.5 Diagrama de Casos de Uso del Sistema

Determinados los requisitos, los actores y casos de usos del sistema se estructura el diagrama de casos de usos del sistema (DCUS), que representa la relación de los actores del sistema con los casos de usos.

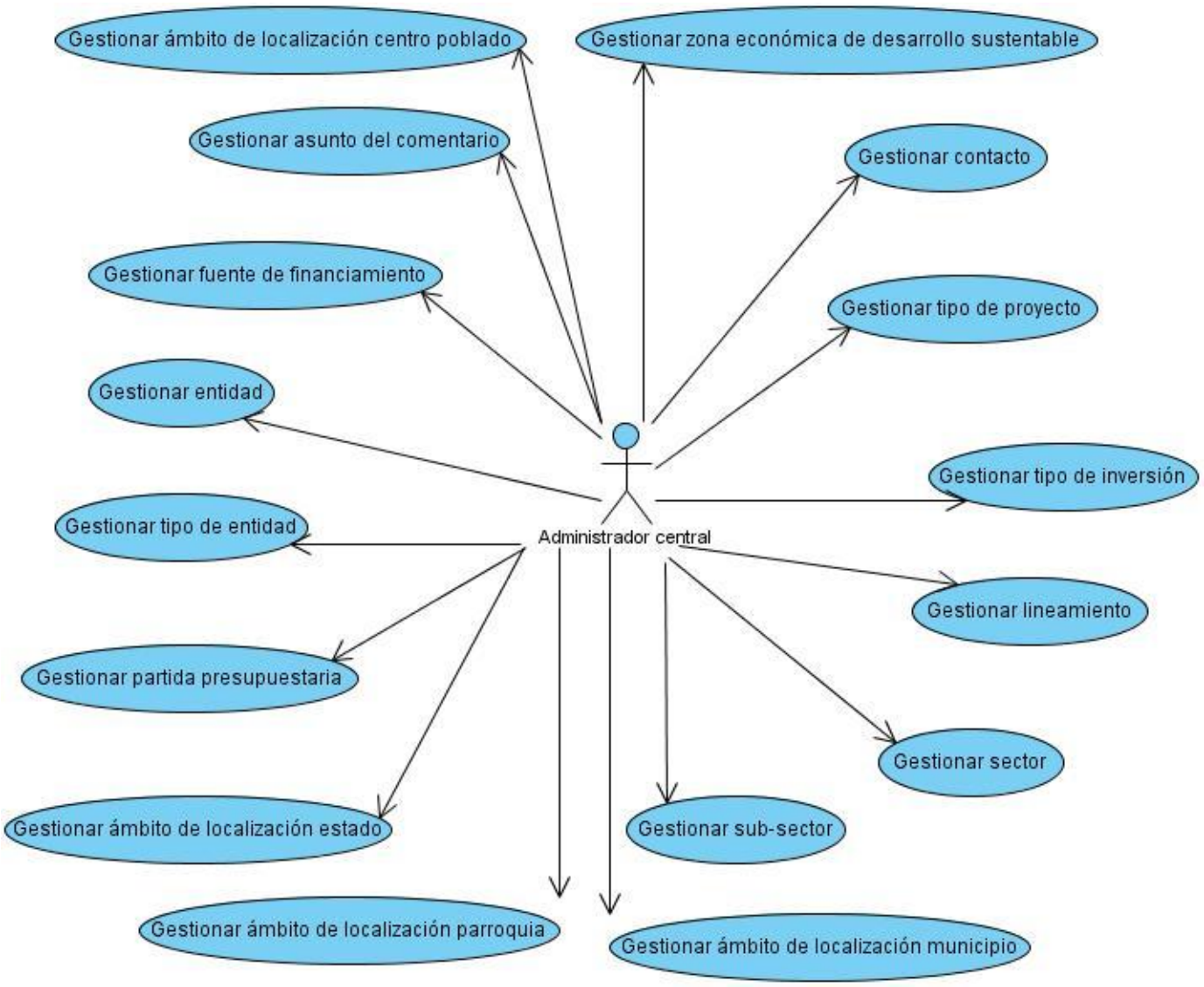


Fig. 15 Diagrama de Casos de Uso del Sistema.

2.6 Descripción de Casos de Uso del Sistema

Un caso de uso tiene una especificación textual que contiene una descripción del flujo de eventos, mostrando como sucede la interacción entre actores y el sistema. A continuación se muestra la descripción de uno de los principales casos de uso del módulo Configuración. Para acceder a la

descripción íntegra de todos los CU, ver el documento Modelo de casos de uso del sistema SINAPSIS, Módulo Configuración. (34)

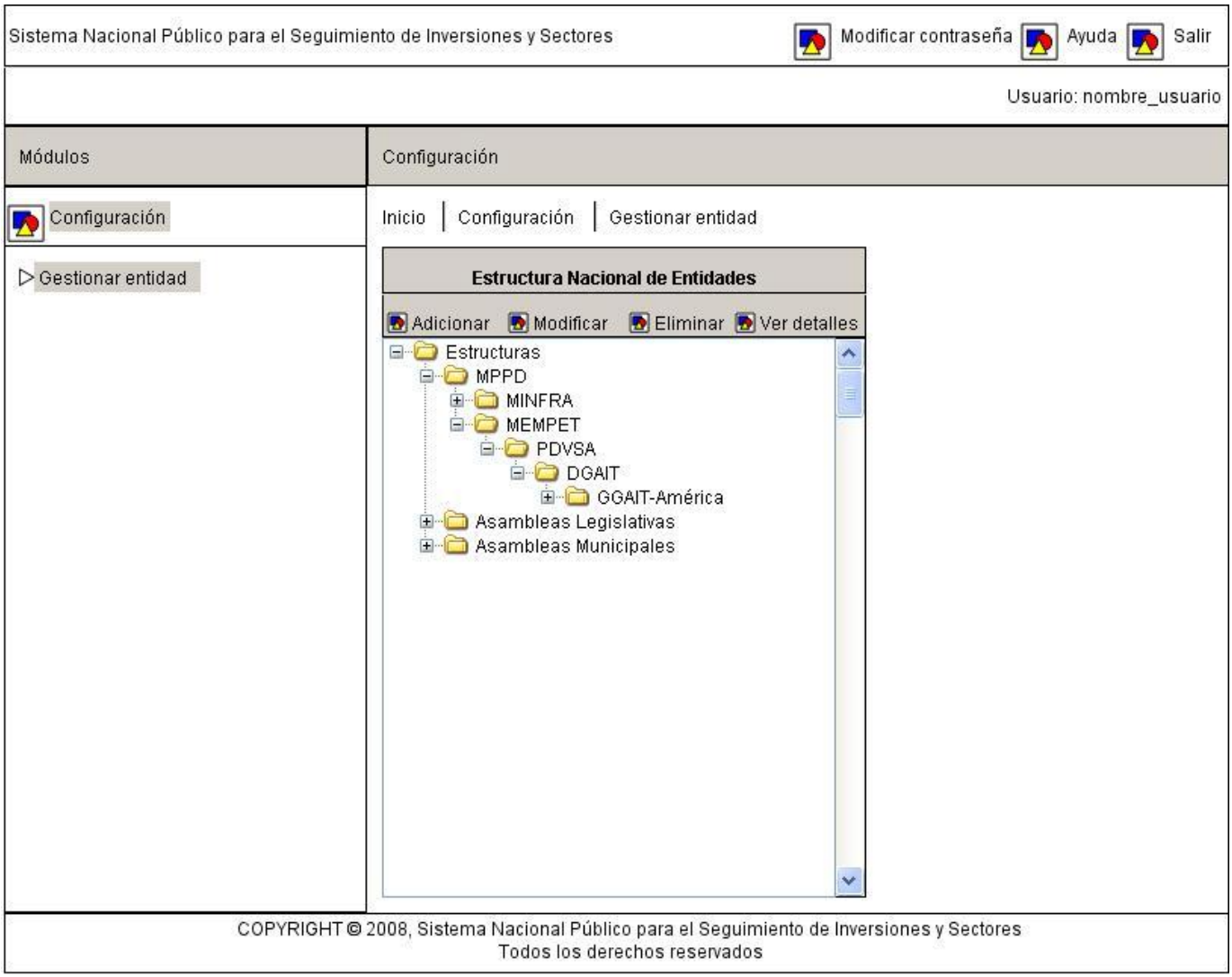
Tabla. 18 Descripción del CU Gestionar entidad.

Caso de Uso:	Gestionar entidad
Actores:	Administrador central
Resumen:	El caso de uso se inicia cuando el Administrador central necesita gestionar una entidad. Consiste en que el Administrador central selecciona la opción de adicionar una nueva entidad y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar una entidad ya sea para modificarla, ver sus detalles o eliminarla. El caso de uso termina con la creación, modificación o eliminación de una entidad.
Precondiciones:	➢ El sistema debe estar instalado y ejecutándose correctamente. ➢ El usuario debe estar autenticado con los permisos necesarios.
Referencias	RF_C.6, RF_C.7, RF_C.8, RF_C.9, RF_C.10, RF_C.78, RF_C.79, RF_C.80
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 central selecciona la opción “Configuración” y dentro de esta la opción “Gestionar entidad”.	2. El sistema muestra una interfaz donde muestra la estructura jerárquica de entidades.
3. El actor Administrador central selecciona una de las siguientes opciones: ➢ Adicionar (Ver sección **“Adicionar entidad”**) ➢ Modificar (una vez que haya seleccionado la entidad que desea modificar, ver sección **“Modificar entidad”**) ➢ Eliminar (una vez que haya seleccionado la entidad que desea eliminar, ver sección **“Eliminar entidad”**)	

➢ Ver detalles (una vez que haya seleccionado la entidad de la que desea ver sus detalles, ver sección **"Ver detalles de la entidad"**)	

Prototipo de Interfaz

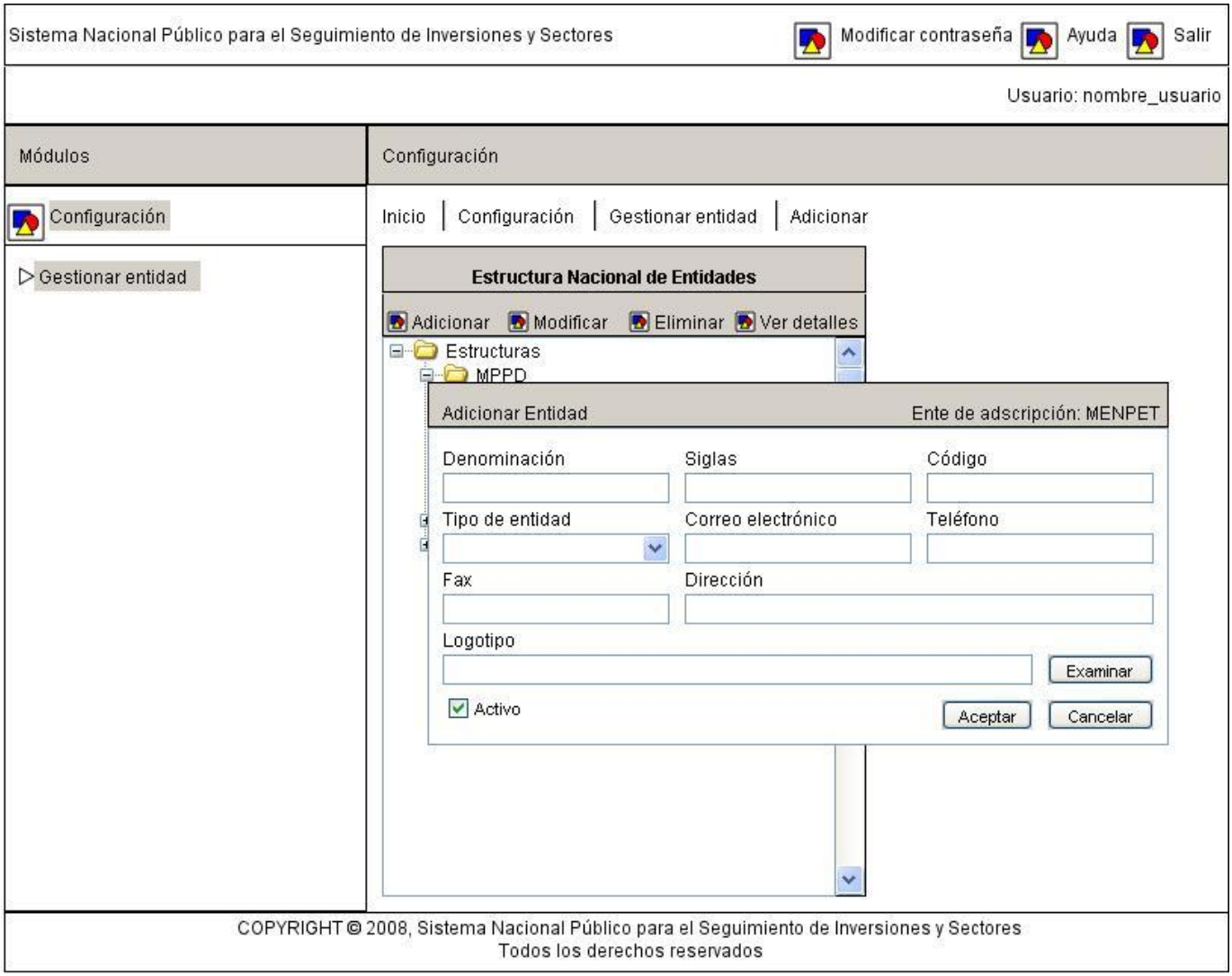


Sección "Adicionar entidad"

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra la interfaz para crear una entidad, solicitando los siguientes datos: ➢ Denominación ➢ Siglas ➢ Código ➢ Tipo de entidad

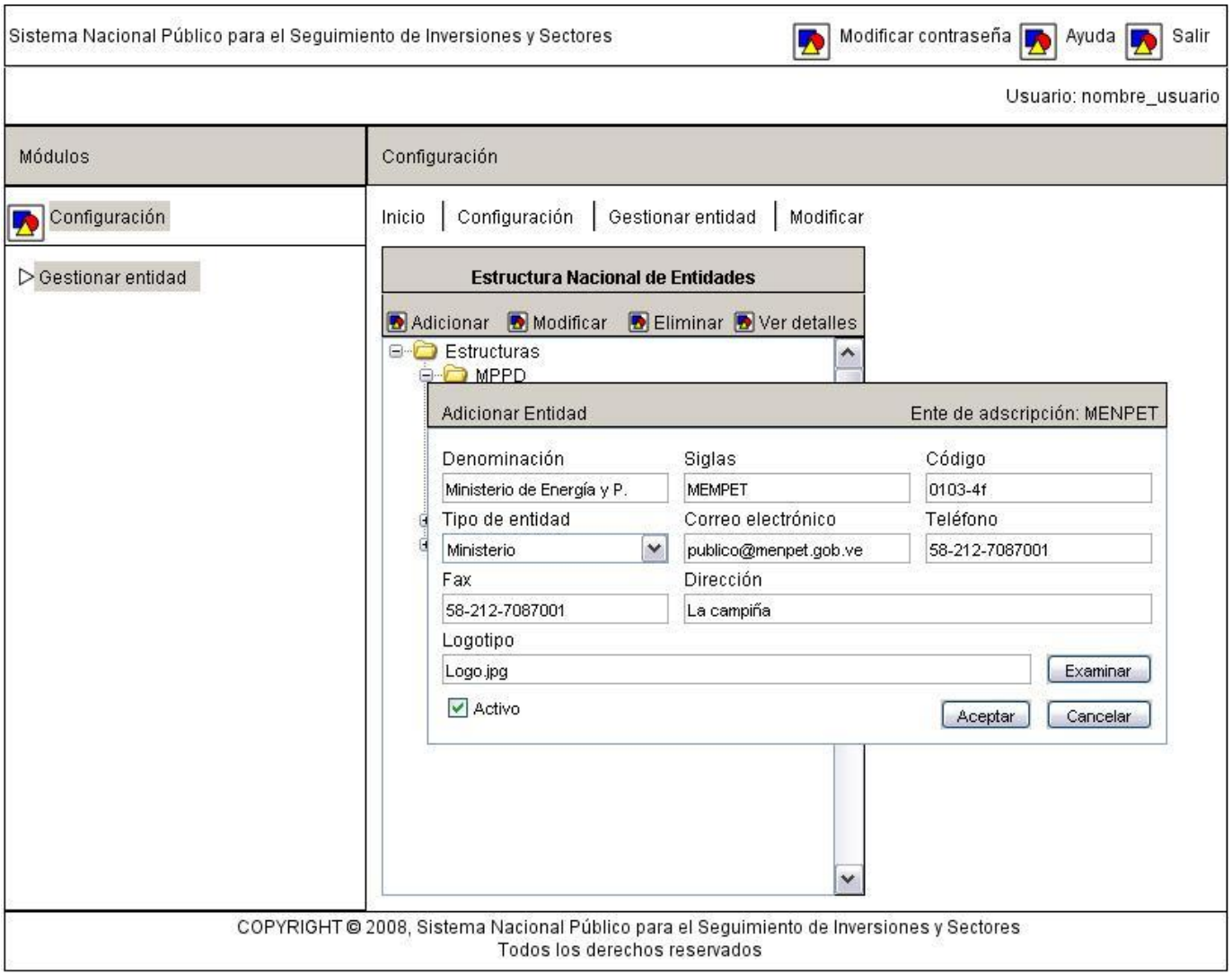
	➢ Correo electrónico ➢ Teléfono ➢ Fax ➢ Dirección ➢ Logotipo ➢ Activo (tipo: booleano)
2. El actor Administrador central introduce los datos correspondientes y selecciona la opción “Aceptar”.	3. El sistema valida que los datos introducidos son correctos y que no hay campos obligatorios vacíos.
	4. El sistema crea la entidad, terminando así el caso de uso.

Prototipo de Interfaz



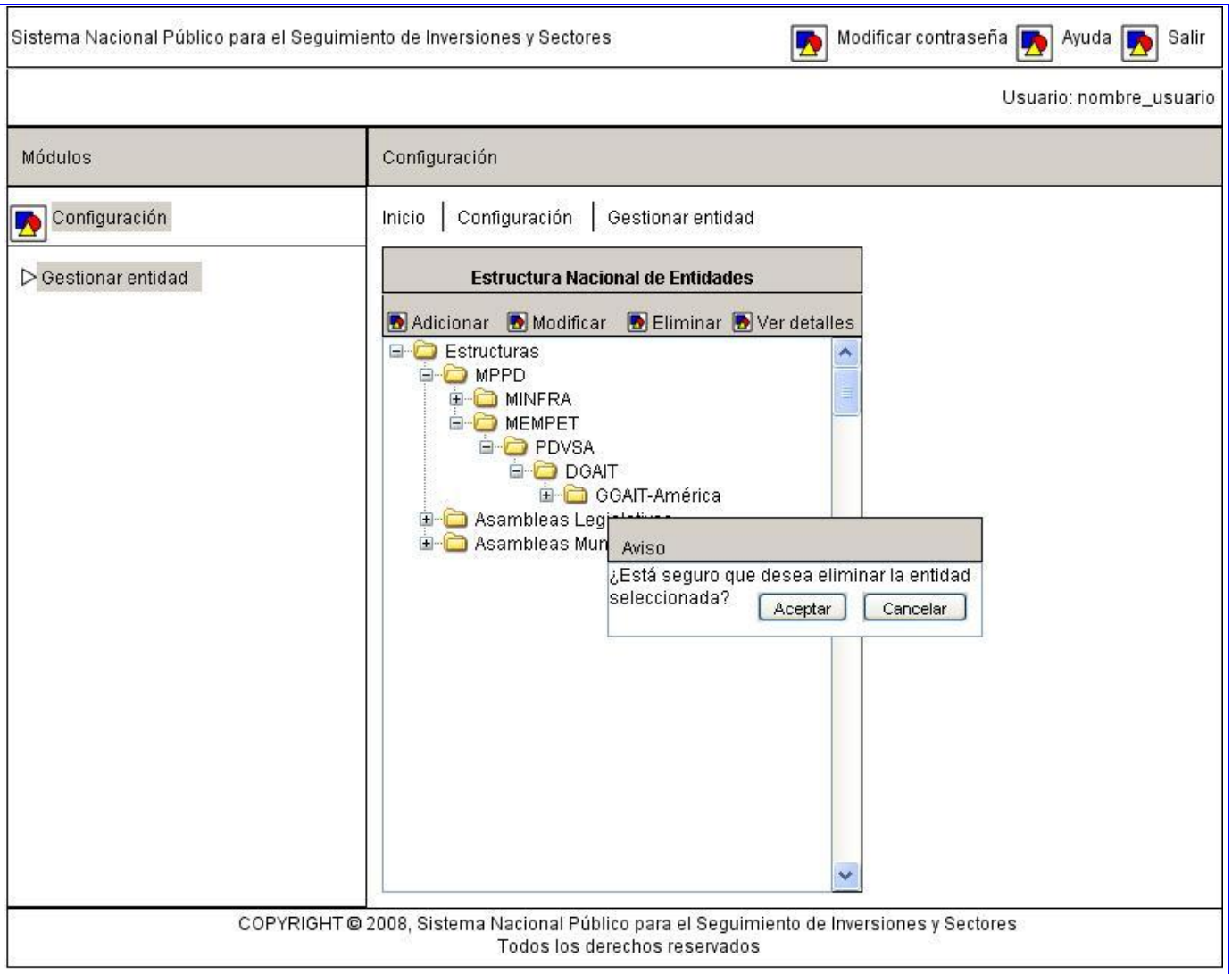
Flujo alterno al paso 2 “Operación cancelada”	
Acción del Actor	**Respuesta del Sistema**
2. a El actor Administrador central selecciona la opción “Cancelar”.	2. b El sistema cancela la operación y regresa al paso 2 del flujo normal de eventos.
Flujo alterno al paso 3 “Datos incorrectos y/o campos vacíos”	
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 “Modificar entidad”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz para modificar los siguientes datos de una entidad: ➢ Denominación ➢ Siglas ➢ Código ➢ Tipo de entidad ➢ Correo electrónico ➢ Teléfono ➢ Fax ➢ Dirección ➢ Logotipo ➢ Activo (tipo: booleano)
2. El actor Administrador central 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 la entidad, 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 central 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

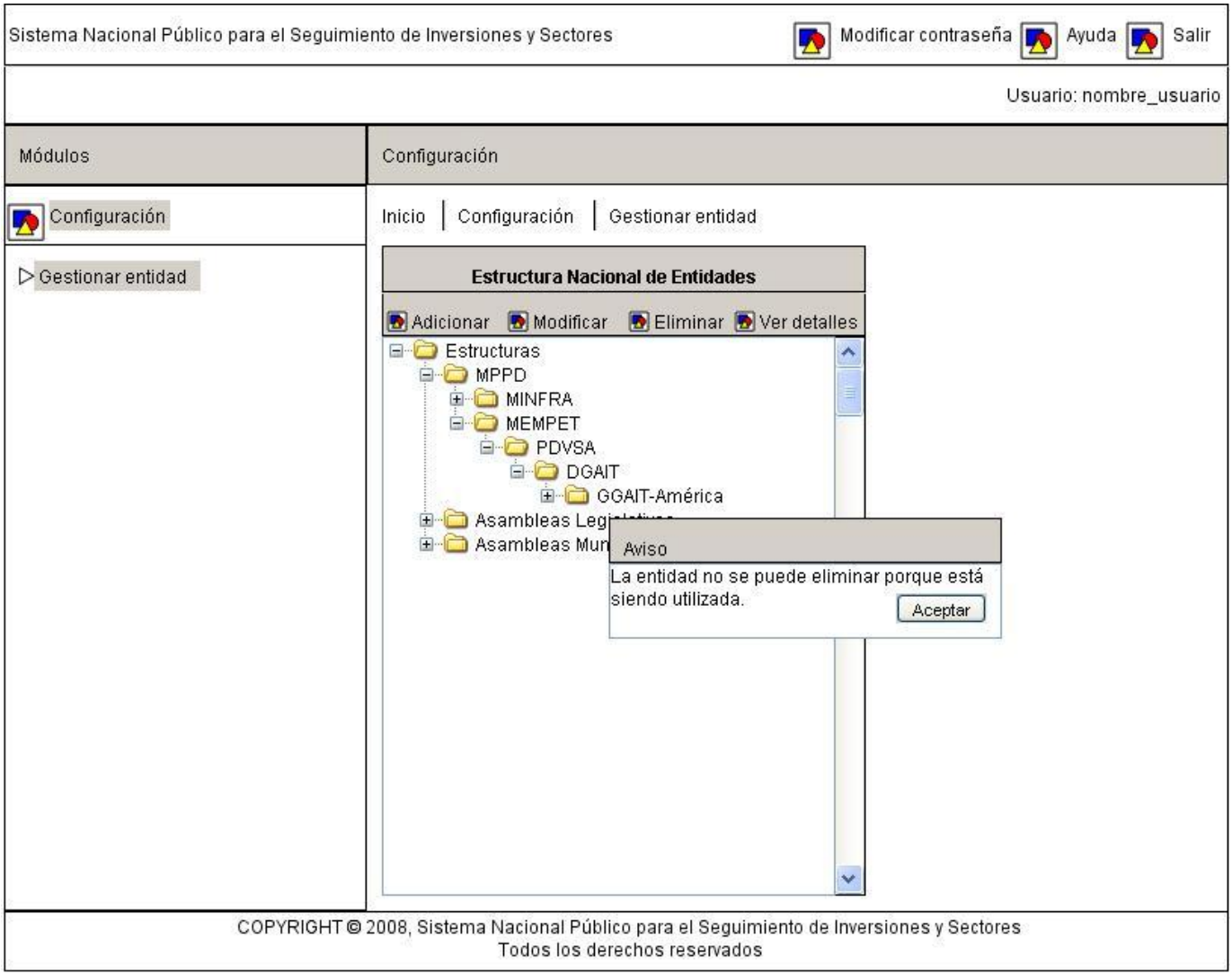
	donde se introdujeron datos incorrectos. Regresa al paso 1 del flujo normal de eventos de esta sección.
Sección "Eliminar entidad"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un mensaje de aviso donde pregunta si está seguro que desea eliminar la entidad seleccionada.
2. El actor Administrador central selecciona la opción "Aceptar".	3. El sistema valida que la entidad puede ser eliminada (que no está siendo utilizada).
	4. El sistema elimina la entidad, 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 central 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 "Entidad utilizada"	
Acción del Actor	**Respuesta del Sistema**
	3. a El sistema valida que la entidad no puede ser eliminada.
	3. b El sistema muestra un mensaje de aviso informando que la entidad no se puede eliminar porque está siendo utilizada.

3. c El actor Administrador central selecciona la opción “Aceptar”.	3. d El sistema regresa al paso 2 de del flujo normal de eventos.

Prototipo de Interfaz

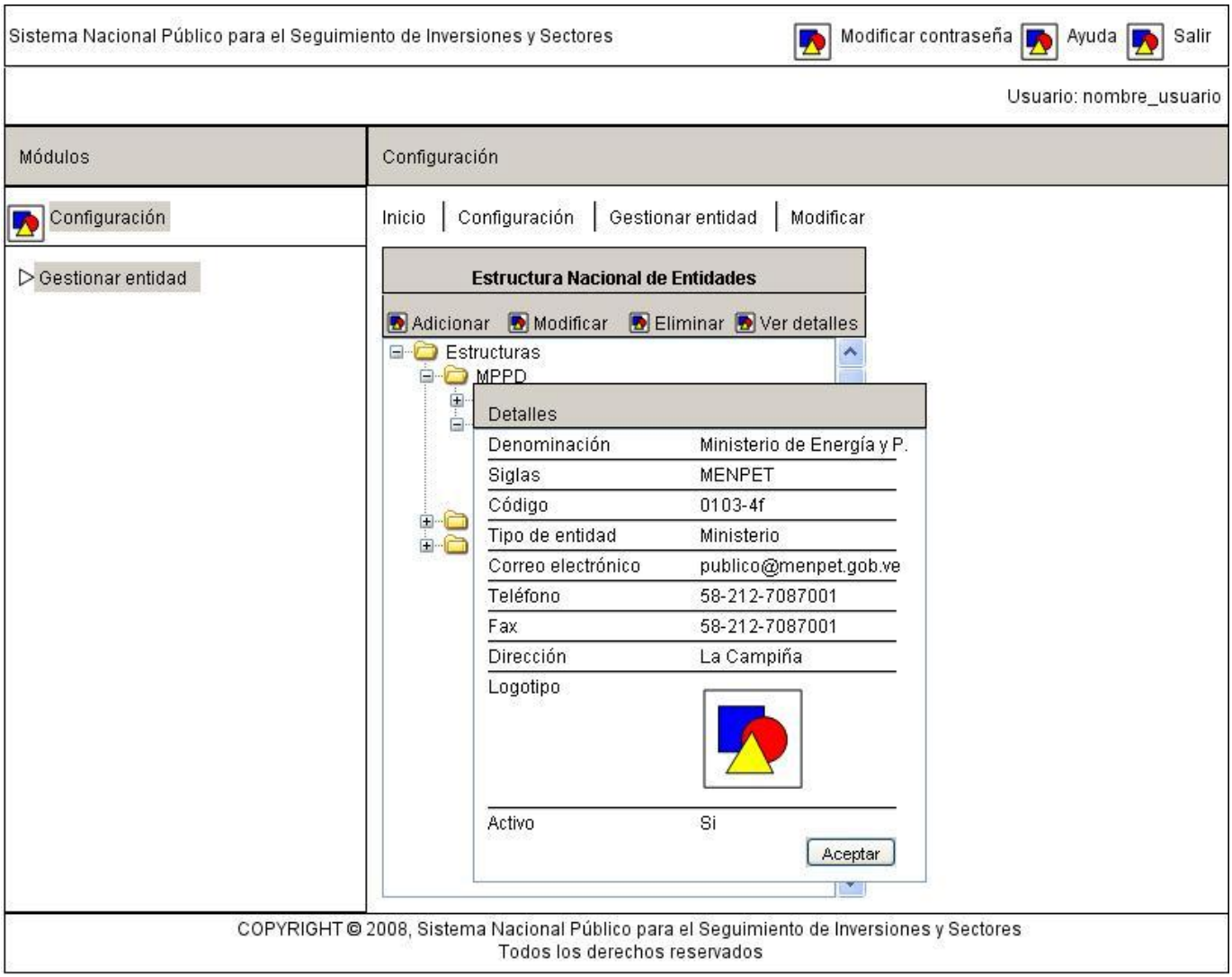


Sección “Ver detalles de la entidad”

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra una interfaz donde se muestran los detalles de la entidad que el Administrador central seleccionó, los campos que se muestran son: ➢ Denominación ➢ Siglas ➢ Código

	➢ Tipo de entidad ➢ Correo electrónico ➢ Teléfono ➢ Fax ➢ Dirección ➢ Logotipo ➢ Activo (tipo: booleano)
2. El actor Administrador selecciona la opción "Aceptar".	3. El sistema regresa al paso 2 del flujo normal de eventos, terminando así el caso de uso.

Prototipo de Interfaz



Pos-condiciones	➢ El sistema queda con una nueva entidad creada. ➢ El sistema queda con una entidad actualizada.

	➢ El sistema queda con una entidad eliminada.

2.7 Subsistemas de Diseño

Los subsistemas de diseño constituyen una forma de estructurar los artefactos que conforman el modelo de diseño en estructuras más independientes, que son las que van a interactuar entre sí para realizar los casos de uso. Los elementos del subsistema acorde con las buenas prácticas, deben tener un alto grado de cohesión y a su vez cada subsistema debe tener bajo acoplamiento con los demás con el objetivo de favorecer la reutilización. (31) A continuación se muestran los subsistemas relacionados con el módulo Configuración del proyecto SINAPSIS.

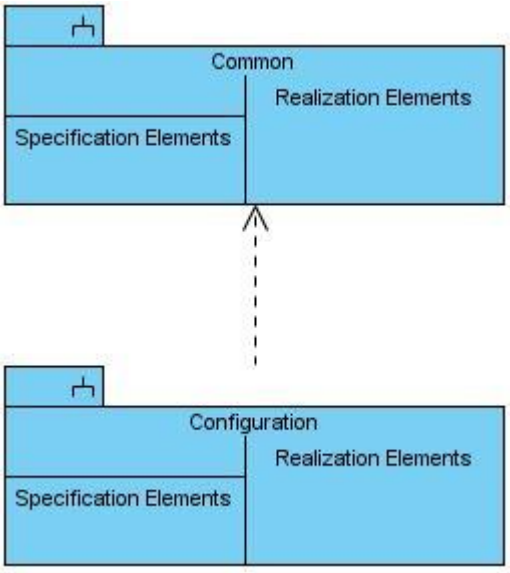


Fig. 16 Subsistemas de Diseño del Módulo Configuración.

2.8 Paquetes de Diseño

Al igual que los paquetes del análisis, los paquetes del diseño son una colección de clases, relaciones, realizaciones de casos de usos y otros paquetes. Son usados para agrupar elementos del modelo de diseño que están relacionados, con el propósito de organizarlos. (31)

Con el objetivo de lograr una mejor organización y especialización de los elementos del diseño se identificaron y definieron los siguientes paquetes para el subsistema de configuración del módulo Configuración del proyecto SINAPSIS.

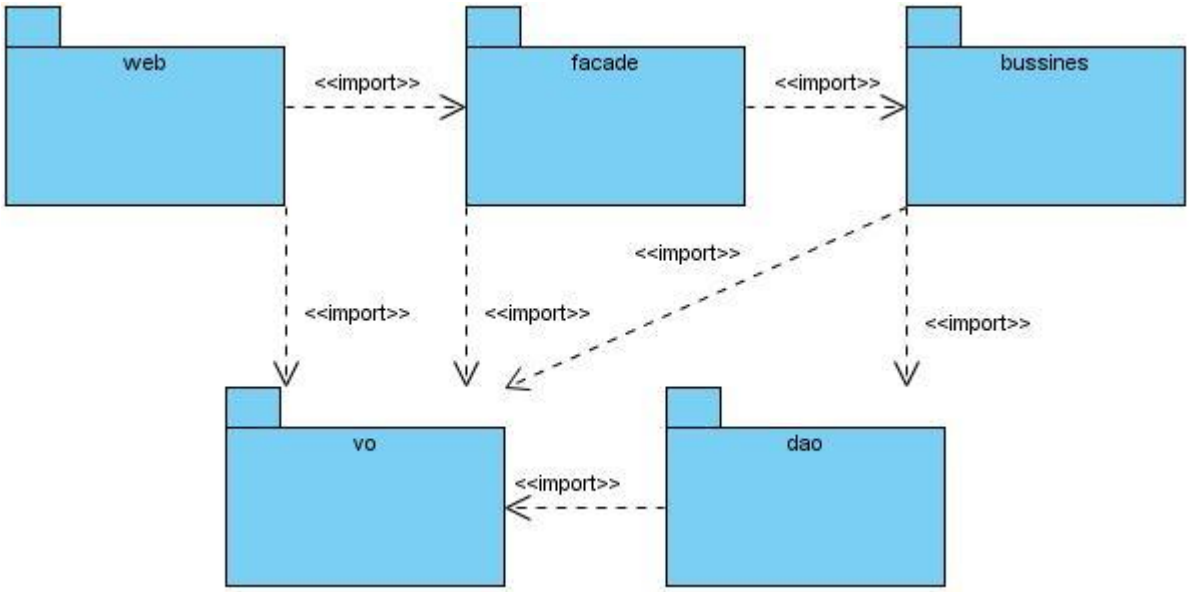


Fig. 17 Paquetes del Diseño del Módulo Configuración.

2.9 Diagramas de Clases del Diseño

Un diagrama de clases de diseño es un diagrama que muestra un conjunto de interfaces, colaboraciones y sus relaciones, se utiliza para modelar principalmente la vista de diseño estática de un sistema. Los diagramas de clases, son importantes, no sólo para visualizar, especificar y documentar modelos estructurales sino también ayudan a construir el sistema a través de la ingeniería directa e inversa de código. (31)

Debido a que el proyecto SINAPSIS debe ser una aplicación web, 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.

A continuación se muestran los diagramas de clases de diseño de uno de los principales casos de uso del módulo Configuración del Proyecto SINAPSIS, el resto de los diagramas de clases del diseño se pueden consultar en el documento referente al modelo de diseño del módulo Configuración del Proyecto SINAPSIS. (35)

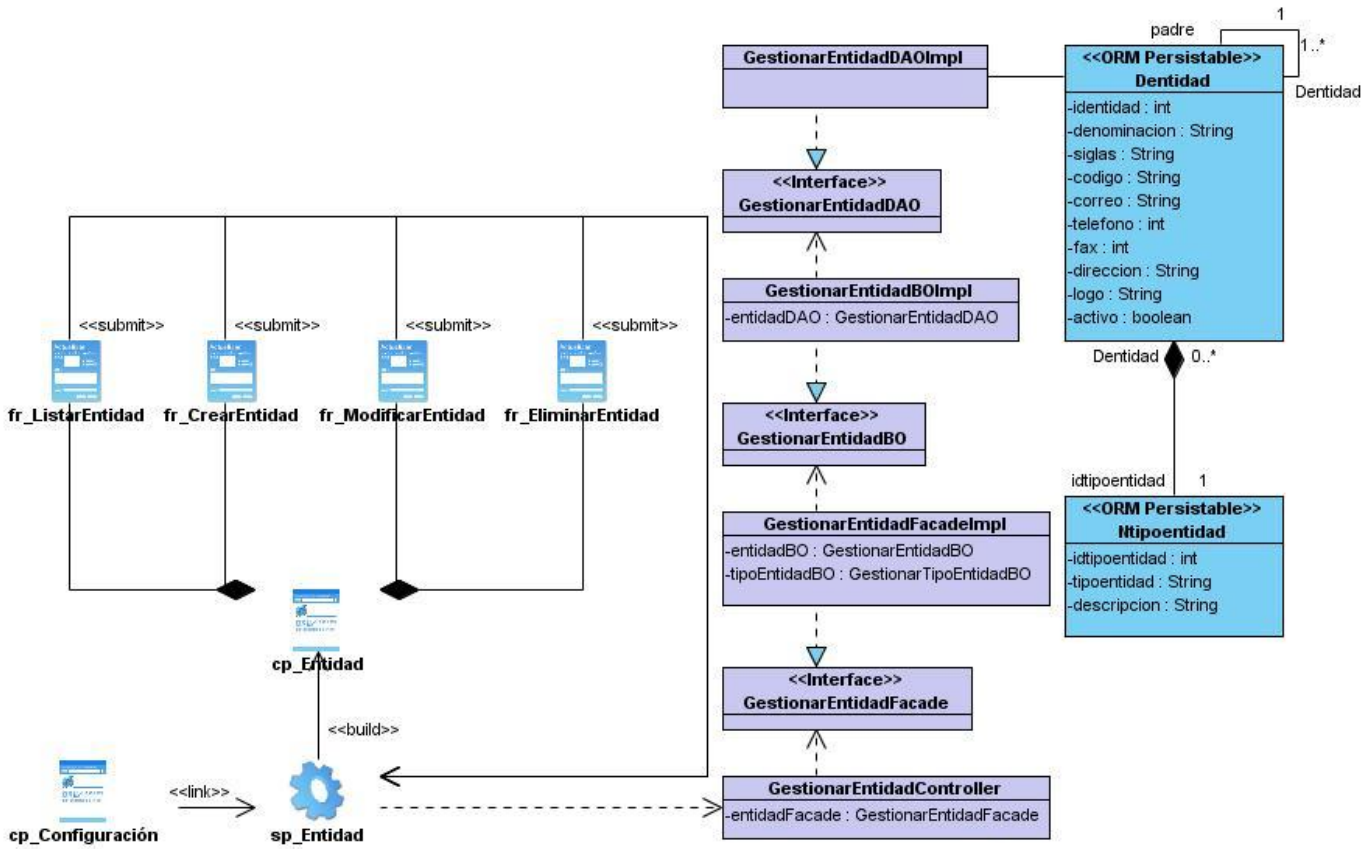


Fig. 18 Diagrama de Clases de Diseño, CU Gestionar Entidad.

2.10 Diagramas de Secuencia

Los diagramas de secuencia persiguen lograr una mejor representación y entendimiento del flujo de eventos para cada caso de uso; estos muestran las interacciones entre objetos, ordenadas en secuencia temporal durante un escenario concreto. Si los casos de uso tienen varios flujos o subflujos distintos, suele ser útil crear un diagrama de secuencia para cada uno de ellos. (31)

A continuación se muestran los diagramas de secuencia pertenecientes a los escenarios de uno de los principales casos de uso del módulo Configuración del proyecto SINAPSIS, el resto de los diagramas de secuencia del diseño se pueden consultar en el documento referente al modelo de diseño del módulo Configuración del proyecto SINAPSIS. (35)

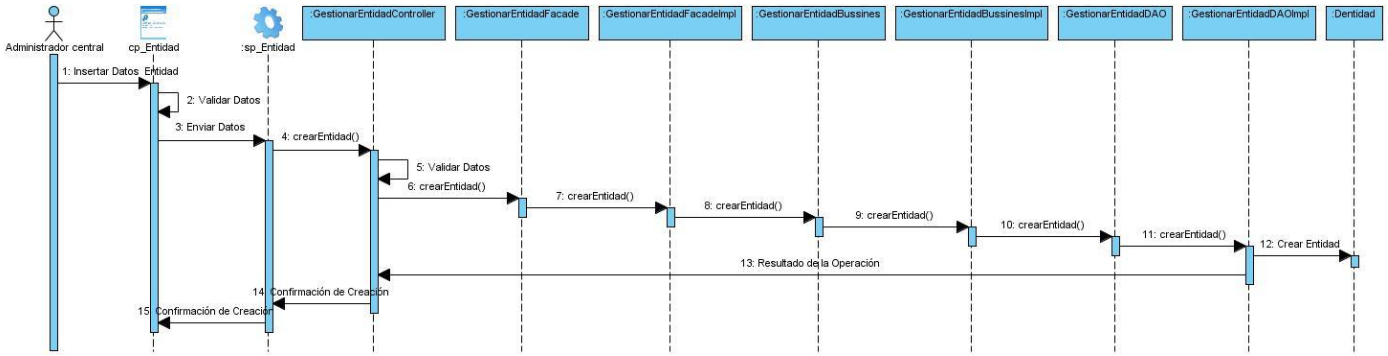


Fig. 19 Diagrama de Secuencia: CU Gestionar Entidad, Sección Crear.

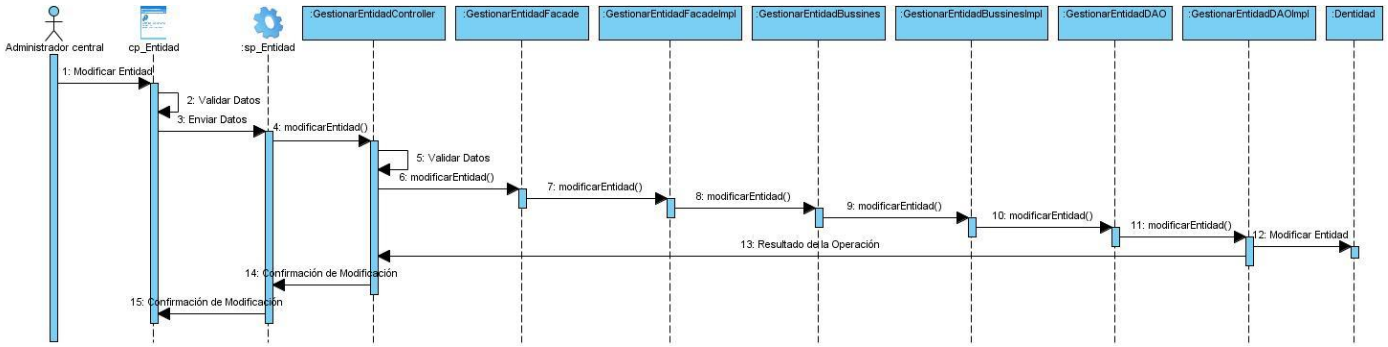


Fig. 20 Diagrama de Secuencia: CU Gestionar Entidad, Sección Modificar.

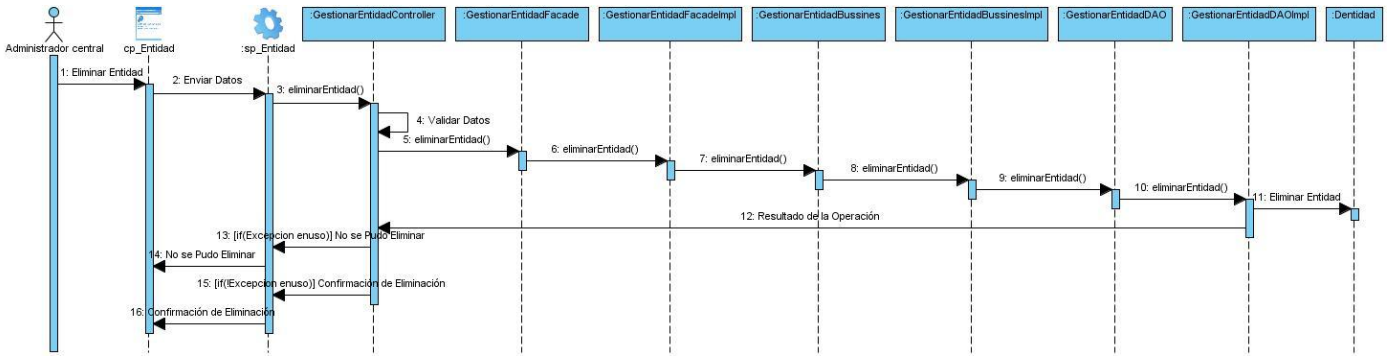


Fig. 21 Diagrama de Secuencia: CU Gestionar Entidad, Sección Eliminar.

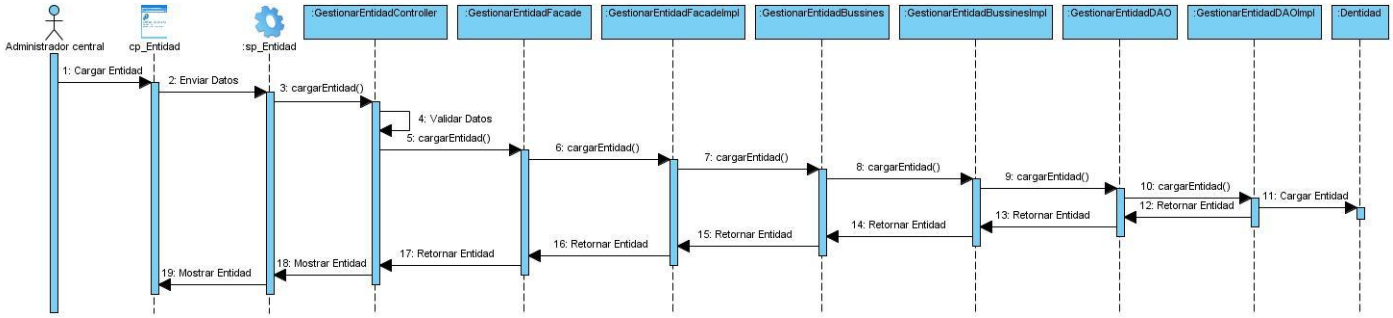


Fig. 22 Diagrama de Secuencia: CU Gestionar Entidad, Sección Ver Detalles.

2.11 Diagrama de Entidad Relación

El modelo de datos describe la representación lógica y física de los datos persistentes utilizados por la aplicación. (31) Con el fin de garantizar la persistencia de los datos se modeló y normalizó el modelo de entidad relación del módulo Configuración del proyecto SINAPSIS, el cual se muestra a continuación.

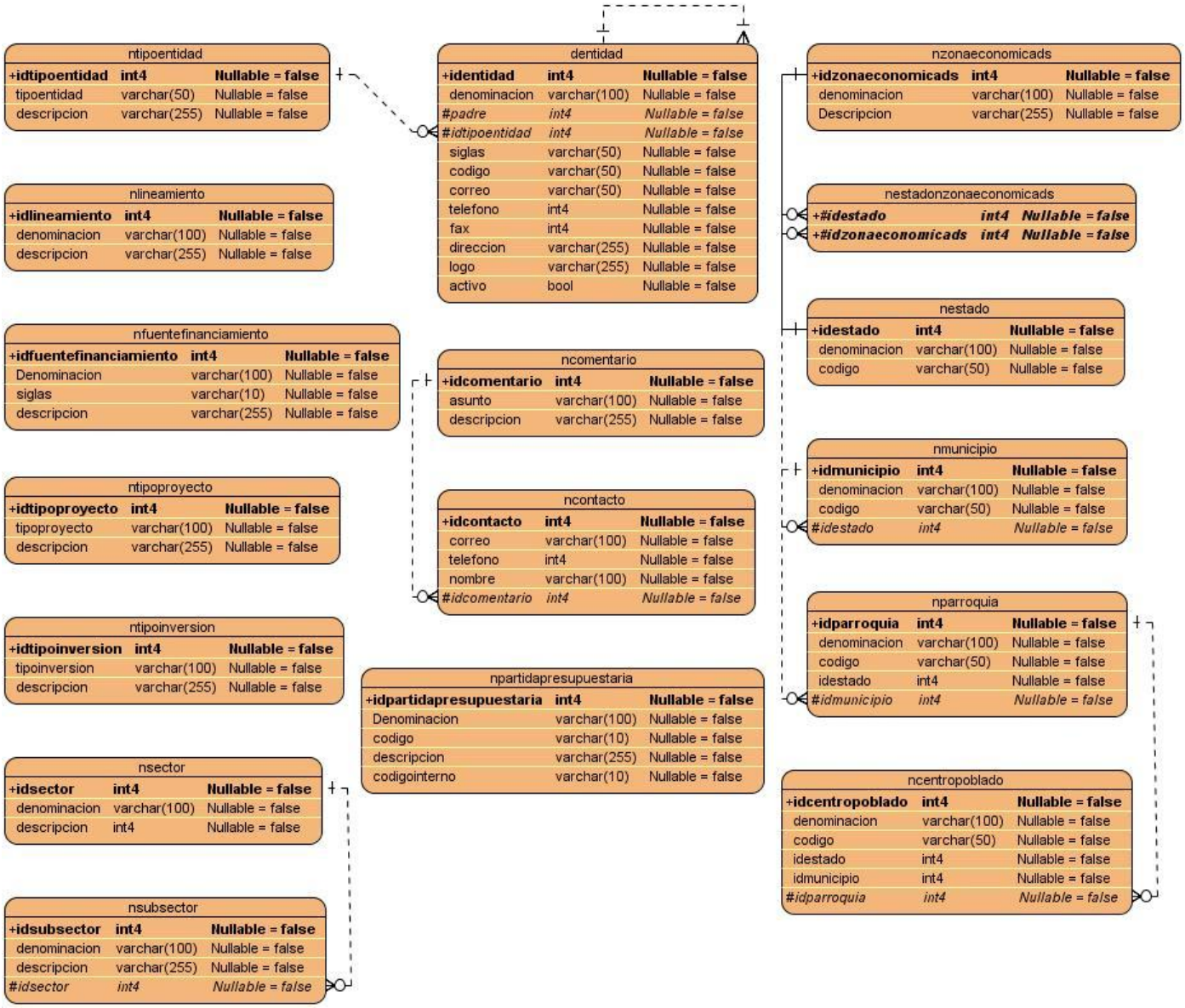


Fig. 23 Modelo Entidad Relación.

2.12 Conclusiones Parciales

En el transcurso de este capítulo partiendo de la interacción con los clientes y aplicando técnicas para la elicitación de requisitos se logró la satisfactoria identificación de los requisitos tanto funcionales como no

funcionales que debe cumplir el sistema. Partiendo de estos se llegó a la realización de los casos de uso lo cual permitió tener una idea más clara de las funcionalidades con las que contará el producto. Se construyó el modelo de diseño dentro del cual se obtuvieron los artefactos que permiten el cumplimiento de los requisitos, siendo esta la entrada del flujo de trabajo de implementación.

Capítulo 3 Análisis de los Resultados

3.1 Introducción

Este capítulo contiene un análisis de los resultados mediante la aplicación de métricas para medir la calidad de cada uno de los principales artefactos obtenidos.

3.2 Validación de la Especificación de Requisitos

Con el objetivo de validar los requisitos y el modelo del sistema, se examinaron las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados se han corregido, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto. Para dar cumplimento a lo antes mencionado se aplicaron listas de chequeo y métricas.

3.2.1 Lista de Chequeo para la Especificación de Requisitos

En la lista de chequeo aplicada para la verificación de la especificación de requisitos las preguntas se agruparon bajo una categoría relativa a una característica de calidad. En cada una de las revisiones se fueron arreglando las no conformidades hasta que la especificación de requisitos satisfizo cada una de las características de calidad contenidas en la lista. Ver Anexo. 1

3.2.2 Métricas para la Especificación de los Requisitos

Para complementar la validación de la especificación de requisitos mediante la aplicación de las listas de chequeo, se aplicaron además las métricas propuestas por Davis y sus colegas, a continuación se muestran los resultados obtenidos.

Número de requisitos en la especificación: $n_r = n_f + n_{nf}$

n_r : Total de requisitos en una especificación.

n_f : Número de requisitos funcionales.

n_{nf} : Número de requisitos no funcionales.

$n_r = 80 + 23 = 103$

Especificidad: $Q_1 = \frac{n_{ui}}{n_r}$

Q_1 : Especificidad de los requisitos.

n_{ui} : Número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.

$Q_1 = \frac{100}{103} = 0.97$

Compleción: $Q_2 = \frac{n_A}{n_A + n_B}$

Q_2 : Compleción de los requisitos.

n_A : Número de requisitos completos.

n_B : Número de requisitos pobremente especificados.

$$Q_2 = \frac{103}{103 + 0} = 1.00$$

Corrección: $Q_3 = \frac{n_c}{n_c + n_{nv}}$

Q_3 : Corrección de los requisitos.

n_c : Número de requisitos que se han validado como correctos.

n_{nv} : Número de requisitos que no se han validado como correctos todavía.

$$Q_3 = \frac{103}{103 + 0} = 1.00$$

Comprensión: $Q_4 = \frac{n_{cu}}{n_r}$

Q_4 : Comprensión de los requisitos.

n_{cu} : Número de requisitos que todos los revisores entienden.

$$Q_4 = \frac{99}{103} = 0.96$$

Capacidad de Verificación: $Q_5 = \frac{n_r}{n_r + \sum_i c(r_i) + \sum_i t(r_i)}$

Q_5 : Capacidad de verificación.

$\sum_i c(r_i)$: Costo para verificar la presencia del requisito i.

$\sum_i t(r_i)$: Tiempo para verificar la presencia del requisito i.

$$Q_5 = \frac{103}{103 + 10.3 + 9.2} = 0.84$$

Consistencia Interna: $Q_6 = \frac{n_u - n_n}{n_u}$

Q_6 : Consistencia interna.

n_u : Número de requisitos especificados.

n_n : Número de requisitos en conflicto con otros requisitos en la especificación.

$$Q_6 = \frac{103 - 0}{103} = 1.00$$

Consistencia Externa: $Q_7 = \frac{n_{EC}}{n_r}$

Q_7 : Consistencia externa.

n_{EC} : Número de requisitos que son consistentes con otros documentos.

$$Q_7 = \frac{103}{103} = 1.00$$

No Redundancia: $Q_8 = \frac{n_f}{n_u}$

n_f : Número de requisitos funcionales.

n_u : Número de requisitos funcionales únicos.

$$Q_8 = \frac{103}{103} = 1.00$$

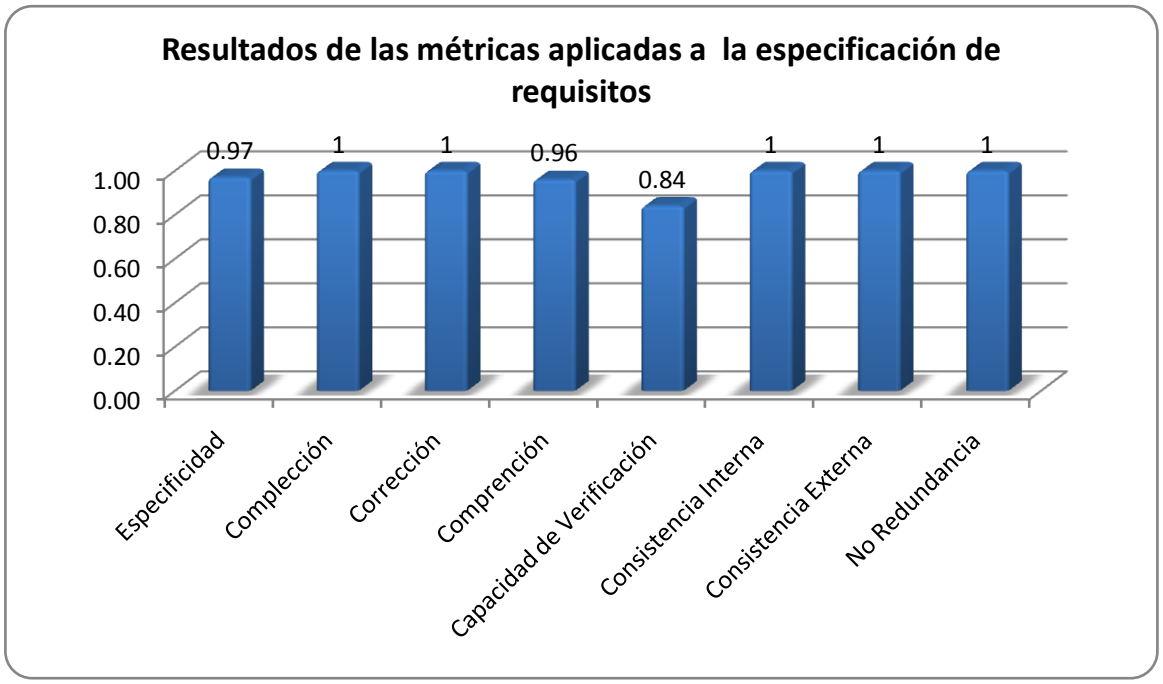


Fig. 24 Resultados de las métricas aplicadas a la especificación de requisitos.

Después de la aplicación de las métricas para la especificación de requisitos y la observación de los resultados obtenidos se puede decir que los mismos son satisfactorios ya que son estables y están por encima de los valores óptimos. Todo esto contribuye al desarrollo exitoso del módulo.

3.2.3 Lista de Chequeo para el Modelo de Sistema

Para la validación del Modelo de sistema se aplicó una lista de chequeo (Anexo. 2), encaminada a verificar la calidad de cada uno de los artefactos contenidos dentro del documento. Se hicieron varias iteraciones corrigiéndose cada una de las no conformidades identificadas.

3.3 Validación del Modelo de Diseño

Para la validación del Modelo de diseño se emplearon varias métricas con el objetivo de evaluar la calidad del mismo en diferentes aspectos tales como arquitectura, componentes y clases contribuyendo de esta forma a la verificación de la correcta realización del mismo.

3.3.1 Métricas de Diseño Arquitectónico

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

$S(i)$: Complejidad estructural

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

$S(i) = 2^2 = 4$

Complejidad de Datos: $D(i) = \frac{V(i)}{f_{out}\,(i)+1}$

$D(i)$: Complejidad en la interfaz interna del módulo Configuración.

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

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

$D(i) = \frac{16}{2+1} = 5.33$

Complejidad del Sistema: $C(i) = S(i) + D(i)$

$C(i)$: Complejidad del sistema.

$C(i) = 4 + 5.33 = 9.33$

Los umbrales definidos para evaluar el resultado de esta métrica se muestran a continuación:

Complejidad	Estructural $S(i)$	Datos $D(i)$	Sistema $C(i)$
No complejo	1, 4. 9, 16 ,25	$<= 7$	$<= 32$
Complejo	36, 49, 64	$> 7\ y\ <= 12$	$> 32\ y\ <= 50$
Muy Complejo	81…	> 12	> 50

Los resultados obtenidos arrojaron que el módulo no es complejo lo que facilitará los procesos de integración y pruebas del módulo Configuración.

3.3.2 Métricas de Diseño a Nivel de Componente

Métricas de cohesión

Tabla. 19 Usabilidad de las clases.

Clases	Usabilidad
GestionarEntidadController	0
GestionarCentroPobladoController	0
GestionarEstadoController	0
GestionarMunicipioController	0
GestionarParroquiaController	0
GestionarTipoEntidadController	0
GestionarEntidadBOImpl	13
GestionarCentroPobladoBOImpl	1
GestionarEstadoBOImpl	5
GestionarMunicipioBOImpl	3
GestionarParroquiaBOImpl	2
GestionarTipoEntidadBOImpl	2
GestionarEntidadDAOImpl	13
GestionarCentroPobladoDAOImpl	1
GestionarEstadoDAOImpl	5
GestionarMunicipioDAOImpl	3
GestionarParroquiaDAOImpl	2
GestionarTipoEntidadDAOImpl	2

La cohesión funcional se determina con dos enfoques:

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

   Se define como:

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

2. Determinando la cohesión funcional débil (CFD):

   Se define como:

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

Adhesivo. Se le llamará adhesivo a un elemento que aparece en dos o más rebanadas.
Súper adhesivo. Se denomina súper adhesivo a un elemento que está en todos los elementos de un módulo.
(i) Se define como la muestra.

Según los datos de las clases analizadas se tiene que:
$número\ de\ elementos\ =\ 66$
$número\ de\ súper\ adhesivos\ (i)\ =\ 0$
$número\ de\ adhesivos\ (i)\ =\ 48$
$CFD\ =\ número\ de\ adhesivos\ (i)\ /\ número\ de\ elementos\ (i)$
$CFD\ =\ 48\ /\ 66$
$CFD\ =\ 0.72$
$CFF\ =\ número\ de\ súper\ adhesivos\ (i)\ /\ número\ de\ elementos\ (i)$
$CFF\ =\ 0\ /\ 66$
$CFF\ =\ 0$

La métrica de Bieman y Ott plantea que mientras más cerca están los valores de CFF y CFD de 1 mayor será la cohesión del módulo. Los resultados demuestran que no hay una cohesión funcional fuerte, pero la relación del número de clases adhesivas con el número total de elementos de la muestra, determinados por la $CFD\ =\ 0.72$, está cercana a 1, lo que demuestra que el diseño de las clases del módulo Configuración posee una cohesión funcional con un 72% de fortaleza.

3.3.3 Métricas Orientadas a Clases

Tamaño de clase (TC)
Para medir el tamaño de clase se tienen en cuenta los siguientes aspectos:
Total de operaciones, ya sean las propias o las heredadas de las clases padres e interfaces que implementen.
Cantidad de atributos, tanto los de ella, como lo de los padres.
Promedio general de los dos anteriores para el sistema completo.

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

Tabla. 20 Tamaño de clases.

Clases	Nº de atributos	Nº de operaciones
GestionarEntidadController	1	8
GestionarCentroPobladoController	1	5
GestionarEstadoController	1	5
GestionarMunicipioController	1	5
GestionarParroquiaController	1	5
GestionarTipoEntidadController	1	5
GestionarEntidadBOImpl	1	8
GestionarCentroPobladoBOImpl	1	5
GestionarEstadoBOImpl	1	5
GestionarMunicipioBOImpl	1	5
GestionarParroquiaBOImpl	1	5
GestionarTipoEntidadBOImpl	1	5
GestionarEntidadDAOImpl	0	9
GestionarCentroPobladoDAOImpl	0	7
GestionarEstadoDAOImpl	0	7
GestionarMunicipioDAOImpl	0	7
GestionarParroquiaDAOImpl	0	7
GestionarTipoEntidadDAOImpl	0	7

Se presentó un **total de 66 clases** para un **promedio de atributos de 0.87** y **un promedio de operaciones de 5.93**.

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

Umbral	Tamaño	Cantidad de Clases
<=20	Pequeño	66
>20<=30	Medio	0
>30	Grande	0

Árbol de Profundidad de Herencia (APH).

Para esta métrica, algunos autores sugieren un umbral de 6 niveles como indicador de un abuso en la herencia en distintos lenguajes de programación.

En el análisis correspondiente se tuvieron en cuenta los diagramas de clases pertenecientes a los paquetes Web, Facade, Bussines y DAO.

Tabla. 21 Profundidad de los paquetes de diseño.

Paquetes	Profundidad
vnz.sinapsis.configuracion.web	1
vnz.sinapsis.configuracion.facade	1
vnz.sinapsis.configuracion.bussines	1
vnz.sinapsis.configuracion.dao	3

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

El certificado de liberación por parte del equipo de calidad de la facultad quince de la Especificación de requisitos, el Modelo de sistema y el Modelo de diseño se encuentra en el Anexo. 3.

3.4 Conclusiones Parciales

En este capítulo se analizaron los resultados obtenidos durante el desarrollo del trabajo, evaluándose los artefactos que componen el Modelo del sistema y los que componen el Modelo del diseño, arribándose a las siguientes conclusiones:

- Las actas de liberación por parte del equipo de calidad de la facultad así como las métricas aplicadas demostraron que tanto los requisitos como la Especificación de los casos de uso del sistema quedaron bien definidos, especificados y sin ambigüedad.
- Los valores de 4 y 5.33 para la complejidad estructural y de datos respectivamente indican que el sistema no es muy complejo.
- El diseño de las clases del módulo Configuración posee una cohesión funcional con un 72% de fortaleza, lo que demuestra que sus elementos están altamente cohesionados.
- El 100% de las clases diseñadas están consideradas como pequeñas, lo que facilitara el proceso de construcción del módulo Configuración.

Las métricas aplicadas y la liberación por el equipo de calidad de la facultad a los artefactos del módulo Configuración validan la calidad del trabajo realizado.

Conclusiones

Al finalizar el presente trabajo se arribó a las siguientes conclusiones:

- Se realizó un estudio del proceso de desarrollo de software, las herramientas a utilizar para modelar, el lenguaje de modelado y la metodología a tener en cuenta, lenguaje de programación, los patrones de casos de uso y de diseño, las métricas para evaluar la calidad de diseño. Permitió obtener la siguiente selección para el desarrollo del trabajo:
  - RUP como metodología de desarrollo de software
  - UML como lenguaje de modelado
  - Visual Paradigm for UML 6.0 como herramienta CASE y como herramienta para la realización de los prototipos de interfaz de usuario.
  - Java como lenguaje de programación.
- Se aplicaron algunas técnicas para capturar los requisitos, tanto funcionales como no funcionales, del sistema. Se tuvieron en cuenta además, algunos patrones para modelar el sistema y el diseño, lo que permitió generar los artefactos deseados para desarrollar un sistema robusto.
- Se evaluó la calidad de los principales artefactos generados: los requisitos, los casos de uso del sistema y el diseño, mediante la aplicación de listas de chequeo y métricas, lo que permitió confirmar la realización de artefactos confiables y con calidad.

Recomendaciones

Con vista al desarrollo exitoso del módulo Configuración, se recomienda:

- Realizar la implementación del módulo Configuración del sistema SINAPSIS.
- Aplicar todas las pruebas pertinentes al producto y que luego sea desplegado para su uso.
- Realizar un seguimiento al software entregado para posibles mejoras o nuevas versiones.

Bibliografía

1. **Internet.** GSInnova. *GSInnova.* [En línea] 2007. [Citado el: 9 de Febrero de 2010.] http://www.rational.com.ar/herramientas/rup.html.

2. —. E-Clases. *E-Clases.* [En línea] [Citado el: 10 de Febrero de 2010.] http://eclases.tripod.com/id11.html.

3. **Bauer, Fritz.** *Software Engineering, Information Processing.* North Holland Publishing Co. : Amsterdam, 1972.

4. **Boehm, B. W.** *Software Engineering, IEEE Fansactions on Computers.* 1976.

5. **Zelkovitz, M. V., Shaw, A. C. y J. D., Gannon.** *Principles of Sopware Enpeering and Design.* Englewoods Clif : Prentice-Hal, 1979.

6. **IEEE.** *Standards Collection: Somare Engineering.* 1993. IEEE Standard 610.12-1990.

7. **Pressman, Roger S.** *Ingeniería del Software. Un Enfoque Practico. Quinta Edición.* Madrid : Mc Graw Hill, 2001.

8. **Young, Ralph R.** *The Requirements Engineering Handbook.* Londres : Atech House, 2004. ISBN: 1-58053-266-7.

9. **Morales Malpica, Ariel E. y Vázquez Thompson, Ornolis.** *Ingeniería de Requerimientos aplicada al módulo Registro y Aprobación del sistema SINAPSIS.* Habana : s.n., 2009.

10. **Durán Toro, Amador y Bernárdez Jiménez, Beatriz.** Metodología para la Elicitación de Requisitos de Sistemas Software. [En línea] Facultad de Informática Universidad de Sevilla, 2000. [Citado el: 11 de Febrero de 2010.] http://www.lsi.us.es/~informes/lsi-2000-10.pdf.

11. **José Escalona, María y Koch, Nora.** Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo. [En línea] Diciembre de 2002. [Citado el: 12 de Febrero de 2010.] http://www.lsi.us.es/docs/informes/LSI-2002-4.pdf.

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

13. **Beck, Kent.** InfoQ. *InfoQ.* [En línea] 19 de Enero de 2009. [Citado el: 8 de Febrero de 2010.] http://www.infoq.com/presentations/Agile-Trends-Kent-Beck.

14. **Canós, José, Letelier, Patricio y Penadés, Carmen.** WillyDev. *WillyDev.* [En línea] 5 de Noviembre de 2008. [Citado el: 9 de Febrero de 2010.] http://www.willydev.net/descargas/prev/TodoAgil.pdf.

15. **Durand, Lilian Martinez y Becerra, Osmany González.** *Análisis y Diseño de un Sistema para la Generación de Reportes.* Habana : s.n., 2008.

16. **Fernández Escribano, Gerardo.** Introducción a Extreme Programming. [En línea] 9 de Diciembre de 2002. [Citado el: 11 de Febrero de 2010.] http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.

17. **Internet.** Hista Internacional S.A. *Hista Internacional S.A.* [En línea] 27 de Febrero de 2007. [Citado el: 9 de Febrero de 2010.] http://www.histaintl.com/servicios/consulting/rup.php.

18. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady.** *El Lenguage Unificado de Modelado. Manual de Referencia.* California : Addison Wesley, 1998.

19. **García, Joaquín.** IngenieroSoftware. *IngenieroSoftware.* [En línea] 7 de Mayo de 2005. [Citado el: 1 de Febrero de 2010.] http://www.ingenierosoftware.com/analisisydiseno/uml.php.

20. **OMG.** Object Management Group - UML. *Object Management Group - UML.* [En línea] 15 de Diciembre de 2009. [Citado el: 6 de Febrero de 2010.] http://www.uml.org/.

21. **Quatrani, Terry.** *Visual Modeling with Rational Rose 2000 and UML.* EEUU : Addison Wesley, 2000. ISBN: 0-201-69961-3.

22. **Internet.** GSInnova. *GSInnova.* [En línea] 2007. [Citado el: 18 de Mayo de 2010.] http://www.rational.com.ar/herramientas/roseenterprise.html.

23. —. Visual Paradigm. *Visual Paradigm.* [En línea] 6 de Febrero de 2010. [Citado el: 6 de Febrero de 2010.] http://www.visual-paradigm.com/.

24. **Figueroa, Pablo.** Universidad de los Andes Departamento de Ingeniería de Sistemas y Computación. [En línea] 22 de Noviembre de 1997. [Citado el: 12 de Febrero de 2010.] http://agamenon.uniandes.edu.co/~pfiguero/soo/Magister_Patrones/intropatrones.html.

25. **Overgaard, Gunnar y Palmkvist, Karin.** *Use Cases: Patterns and Blueprints (Hardcover).* s.l. : Addison-Wesley Professional , 2004. ISBN-10: 0131451340 & ISBN-13: 978-0131451346 .

26. **García, Joaquín.** IngenieroSoftware. *IngenieroSoftware.* [En línea] 27 de Mayo de 2005. [Citado el: 8 de Febrero de 2010.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

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

28. **Internet.** Maestros del web. *Maestros del web.* [En línea] [Citado el: 2010 de Mayo de 25.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

29. —. oHost. *oHost.* [En línea] 2009. [Citado el: 2010 de Mayo de 25.] http://www.ohost.com.ar/glosario.php.

30. **Rational Software Corporation.** *Ayuda en Línea del Rational Rose.* 2003.

31. **Fuerte Infante, Minardo y Pérez Luis, Danis Diosdado.** *Especificación de Requisitos, Módulo Configuración, Proyecto SINAPSIS.* Ciudad de la habana : s.n., 2010.

32. **Pérez Luis, Danis Diosdado y Fuerte Infante, Minardo.** *Especificación de Requisitos No Funcionales, Módulo Configuración, Proyecto SINAPSIS.* Ciudad de la Habana : s.n., 2010.

33. **Fuerte Infante, Minardo y Pérez Luis, Danis Diosdado.** *Modelo del Sistema, Módulo Configuración, proyecto SINAPSIS.* Ciudad de la Habana : s.n., 2010.

34. **Pérez Luis, Danis Diosdado y Fuerte Infante, Minardo.** *Modelo de Diseño, Módulo Configuración, Proyecto SINAPSIS.* Ciudad de la Habana : s.n., 2010.

35. **Fuerte Infante, Minardo y Pérez Luis, Danis Diosdado.** *Listas de Chequeo.* Ciudad de la Habana : s.n., 2010.

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

37. **Letelier.** Portal de Desarrollo de Software. *Portal de Desarrollo de Software.* [En línea] 25 de Septiembre de 2006. [Citado el: 11 de Febrero de 2010.] https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducción%20a%20RUP.doc.

38. **Universidad de la Republica Uruguay.** Facultad de Ingeniería. [En línea] [Citado el: 13 de Febrero de 2010.] http://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/experiencia2002/ModsGX/modelo/roles/ana.htm.

39. **Internet.** Maestros del web. *Maestros del web.* [En línea] [Citado el: 2010 de Mayo de 25.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

40. **Villaverde Mesa, Pablo.** Recursos para la programacion en ASP. *Recursos para la programacion en ASP.* [En línea] 2006-2007. [Citado el: 2010 de Mayo de 10.] http://tgp0607.awardspace.com/Recursos_ASP.pdf.

Anexos

Anexo. 1 Lista de chequeo para la Especificación de requisitos.

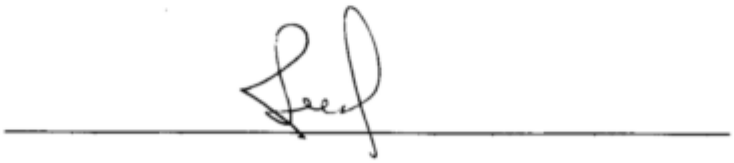
Anexo. 2 Lista de chequeo para el Modelo de sistema. **(36)**

Anexo. 3 Acta de Liberación de Artefactos.

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 No Funcionales, Especificación de Requisitos, Modelo de Sistema y Modelo de diseño del módulo Configuración del proyecto Sinapsis del Centro CEGEL de la Facultad 15 y haberse detectado 6 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

Anexo. 4 Acta de Aceptación de SINAPSIS.

PDVSA

Caracas, 07 de Diciembre de 2009.

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

Estimado: **Juan Carlos Montané Izaguirre**

Después de analizado los documentos:

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

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

PDVSA

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

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

Atentamente,

Saludos,

07/12/09

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

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

Glosario de Términos

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

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

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

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

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

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

IEEE: Corresponde a las siglas de The Institute of Electrical and Electronics Engineers (el Instituto de Ingenieros Eléctricos y Electrónicos), es la asociación técnica y profesional, sin fines de lucro, más grande del mundo formada por profesionales de todas las disciplinas de la ingeniería. Su trabajo es promover la creatividad, el desarrollo y la integración, compartir y aplicar los avances en las tecnologías de la información, electrónica y ciencias en general para beneficio de la humanidad y de los mismos profesionales.

Ingeniería de Requisitos: es el ”uso sistemático de procedimientos, técnicas, lenguajes y herramientas para obtener con un coste reducido el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo de un sistema que satisfaga las necesidades del usuario”.

MPPD: Ministerio del Poder Popular para la Planificación y Desarrollo.

OMG: Object Management Group es una asociación sin fines de lucro formada por grandes corporaciones, muchas de ellas de la industria del software, como IBM, Apple, Sun Microsystems y HP. Se encarga de la definición y el mantenimiento de estándares para aplicaciones de la industria de la computación, como UML, CORBA y otros.

PDF: acrónimo del inglés Portable Document Format (formato de documento portátil), es un formato de almacenamiento de documentos, de tipo compuesto (imagen vectorial, mapa de bits y texto). Está

especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la presentación final del documento, determinando todos los detalles de cómo va a quedar, no requiriéndose procesos anteriores de ajuste ni de maquetación.

Paquetes de diseño: Es una colección de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes que estén de alguna forma relacionados. No definen un comportamiento o semántica bien definida.

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

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

Release: Producto final preparado para lanzarse como versión definitiva a menos que aparezcan errores que lo impidan. Implementa todas las funciones del diseño y se encuentra libre de cualquier error que suponga un punto muerto en el desarrollo.

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

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

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

Subsistemas de diseño: Representan una separación de aspectos lógicos de diseño. Constituyen una forma de organizar los artefactos del Modelo de diseño en piezas más manejables. Puede contener clases del diseño, realizaciones de casos de uso, interfaces y otros subsistemas.

UML: Acrónimo del inglés Unified Modeling Language (Lenguaje Unificado de Modelado). Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el