UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 15

Requerimientos y diseño del módulo de Ficha Planes de Personal del Sistema Nacional Público para el Seguimiento de las Inversiones y Sectores

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

Autores: Yudelkys Carrasco Ramírez

José Angel Martínez García

Tutores: Ing. Ariel Eduardo Morales Malpica

Ing. Martín Villalón Cruzata

Ciudad de La Habana, 2009-2010

DECLARACIÓN DE AUTORÍA

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

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

Yudelkys Carrasco Ramírez	José Angel Martínez García
_____	_____
(Autor)	(Autor)
Ing. Ariel Eduardo Morales Malpica	Ing. Martin Villalón Cruzata
_____	_____
(Tutor)	(Tutor)

AGRADECIMIENTOS

A nuestros familiares por contribuir infinitamente en la realización de nuestros sueños. ¡Los queremos!

A nuestro Comandante en Jefe Fidel Castro Ruz por habernos dado la posibilidad de estudiar en una "Universidad del Futuro".

A nuestros tutores por ayudarnos en el desarrollo del presente trabajo.

A Aldo Gutiérrez Rivera por su apoyo.

A los profesores, compañeros y amigos que ayudaron de una u otra forma al desarrollo de esta tesis.

A todos ustedes, gracias.

Muchas Gracias

DEDICATORIA

En este día tan especial para mí, dedico este trabajo a todos los que de una forma u otra estuvieron siempre conmigo:

A mi mamita querida, por haberme educado con devoción y esmero, por confiar en mí, por guiarme siempre por el camino correcto y por ser la mejor madre del mundo. Te quiero infinitamente…

A Yuliet y Yusmary más que mis hermanas preciosas, mis segundas madres por su inmenso cariño, su confianza, su apoyo y sus consejos. Las quiero con la vida.

A mi hermanito del alma, gracias por estar siempre conmigo.

A mi padre, por su preocupación y apoyo.

A mis sobrinitos July y Alex, los quiero mucho.

A mis amigas Maritza, Marlén, Alianny, Bárbara y de manera muy especial a mi amiga Danise, gracias por haber estado ahí siempre, en los buenos y malos momentos de mi vida universitaria.

A mis compañeros de aula de todos estos años con los cuales compartí muchas alegrías y a los cuales recordare siempre.

Yudelkys Carrasco Ramírez

DEDICATORIA

Quisiera dedicar este trabajo a todas las personas que siempre me han apoyado en los momentos buenos y también en los momentos más difíciles.

A mi madre, guía y apoyo. Siempre exigente buscando dé lo mejor de mí. Gracias. Te necesito mucho y te quiero por sobre todas las cosas.

A mi padre, por su apoyo incondicional. También te necesito. Cuida mucho tu salud. Te quiero.

A mis tíos y abuelos, por su cariño y su influencia en mi formación.

A Dayamí, porque tu consejo nunca ha faltado en los momentos difíciles. Me has enseñado mucho y tu apoyo ha sido muy importante para mí. Eres alguien especial a quien aprecio mucho.

A mi amigo de la infancia Julito. Hemos crecido juntos y siempre nos hemos ayudado. Ya logramos lo que un día nos propusimos: somos universitarios.

A mis compañeros de aula con los cuales he compartido en esta universidad mágica. Los recordaré siempre.

José Angel Martínez García

RESUMEN

El Ministerio del Poder Popular para la Planificación y Desarrollo de la República Bolivariana de Venezuela (MPPPD), es el encargado de la regulación, formulación y seguimiento de las políticas de planificación y desarrollo institucional, entre otras funciones, siendo un organismo de vital importancia para el desarrollo de este país y del proceso revolucionario de cambios que se está realizando.

El MPPPD utiliza un sistema informático denominado Nueva Etapa, para llevar a cabo la gestión del presupuesto por proyectos. Nueva Etapa ha presentado una serie de problemas que dificultan el proceso de gestión del presupuesto de los proyectos, lo que implica que las estimaciones de costos y tiempos sean irreales.

En septiembre del 2009 se decidió crear un nuevo sistema que reemplazara a Nueva Etapa, el cual se denominó SINAPSIS. Este sistema está dividido en 8 módulos, entre ellos el llamado Ficha Planes de Personal, concebido para minimizar la complejidad y heterogeneidad del proceso de control de recursos humanos y materiales, para la ejecución de los proyectos en el sistema SINAPSIS.

En el presente trabajo se realiza la ingeniería de requerimientos y el diseño del módulo Ficha Planes de Personal, con el objetivo de obtener un sistema bien especificado. Para ello se identificaron, analizaron y especificaron los requerimientos y casos de uso del sistema. Además, se generaron los artefactos correspondientes al modelo de diseño: subsistemas, paquetes, modelo de datos, diagramas de clases y diagramas de secuencia. Finalmente, los artefactos de software obtenidos se validan mediante métricas, tanto en el modelo de casos de uso del sistema como en el modelo de diseño en función de la calidad.

Palabras claves

Requerimientos, casos de uso del sistema, modelo de diseño, artefactos de software y métricas.

ÍNDICE DE FIGURAS

ÍNDICE

INTRODUCCIÓN

El Ministerio del Poder Popular para la Planificación y Desarrollo de la República Bolivariana de Venezuela (MPPPD) es el encargado de la regulación, formulación y seguimiento de las políticas de planificación y desarrollo institucional, además de la formulación del Plan de Inversiones Públicas, y de proponer los lineamientos de la planificación del Estado y de la planificación física y espacial a escala nacional entre otras funciones, siendo un organismo de vital importancia para el desarrollo de este país y del proceso revolucionario de cambios que se está realizando.

El Ministerio del Poder Popular para la Planificación y Desarrollo ha reconocido la deficiencia existente en el área de la planificación del sector público venezolano, así como el ineficiente manejo de los recursos asignados por el Estado, debido a la falta de información y la presencia heterogénea de diferentes sistemas informáticos, lo cual dificulta obtener información real y sistemática para el control de los recursos asignados a los diferentes organismos del sector público, para el desarrollo de proyectos con carácter social y económico.

Para llevar a cabo la gestión del presupuesto por proyectos, a principios del 2006 se creó una comisión integrada por el Ministerio del Poder Popular para la Planificación y Desarrollo, el Ministerio del Poder Popular para Economía y Finanzas, la Vicepresidencia de la República y Automática-Informática-Telecomunicaciones de Petróleos de Venezuela S.A. Esta comisión decidió implantar un sistema informático que facilitara la gestión del presupuesto anual por proyectos de la nación, denominado Nueva Etapa, siendo Automática-Informática-Telecomunicaciones de Petróleos de Venezuela S.A. el organismo responsable de la implementación del sistema.

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, ya que no existe un proceso homogéneo de registro y aprobación 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.

En septiembre del 2009 se creó una nueva comisión compuesta por el Ministerio del Poder Popular para la Planificación y Desarrollo, el Ministerio del Poder Popular para Economía y Finanzas, Automática-Informática-Telecomunicaciones de Petróleos de Venezuela S.A, el Centro Nacional de Tecnologías de Información, ALBET S.A en

representación de la Universidad de las Ciencias Informáticas y la Vicepresidencia de la República. Esta comisión analizó los problemas de Nueva Etapa y definió los procesos que debían ser mejorados quedando responsables, para el aseguramiento informático y tecnológico, el Centro Nacional de Tecnologías de Información, y para la implementación del Sistema Nacional Público para el Seguimiento de Inversiones y Sectores (SINAPSIS), la Universidad de las Ciencias Informáticas, entidad que ha desarrollado un número considerable de sistemas informáticos para diversos organismos y que ha obtenido importantes resultados para el estado cubano en materia de informatización del país y en la prestación de servicios asociados en el exterior.

Con el desarrollo de SINAPSIS se pretende lograr un producto altamente configurable y modular de alcance nacional, para la gestión integral de los procesos de planificación del estado venezolano. Este sistema se divide en 8 módulos, entre ellos Ficha Planes de Personal, concebido para minimizar la complejidad y heterogeneidad del proceso de control de recursos humanos y materiales para la ejecución de los proyectos en el sistema SINAPSIS, el cual requiere un control del capital humano asignado a los proyectos. De ahí la importancia de digitalizar y controlar la Ficha Planes de Personal, para lo que es necesario obtener un conjunto de artefactos de software que permita un desarrollo posterior del sistema, una vez que se comprenda el proceso completamente.

El conjunto de artefactos identificados y especificados permitirá a clientes, usuarios y desarrolladores, una mejor comprensión de las funcionalidades y características que estarán presentes en el módulo de Fichas Planes de Personal del sistema SINAPSIS.

Teniendo en cuenta lo antes expuesto surge el siguiente **problema científico:** los artefactos de software actuales del módulo Ficha Planes de Personal del sistema SINAPSIS, no garantizan el comienzo de la implementación del módulo, por lo que nuestra tesis se propuso como **objeto de estudio,** el Proceso de Desarrollo de Software, y como **campo de acción,** la Ingeniería de Requerimientos y Diseño. De esta forma, el **objetivo general** de este trabajo es generar los artefactos de software necesarios para el módulo Ficha Planes de Personal del sistema SINAPSIS, que permitan el comienzo de la implementación del módulo. Utilizando como referencia el objetivo general y el problema científico se ha obtenido como resultado la siguiente **hipótesis**: Si se generan los artefactos necesarios para el módulo de Ficha Planes de Personal del sistema SINAPSIS, se garantizará el comienzo de la implementación del mismo.

Para lograr el objetivo se dará cumplimiento a las siguientes **tareas de la investigación**:

- Estudio del proceso de desarrollo de software.
- Estudio de las metodologías y herramientas para la ingeniería de requerimientos y el diseño.

- Identificación de los requerimientos funcionales y no funcionales del sistema.
- Especificación de los requerimientos.
- Diseño de los elementos del sistema.
- Aplicación de métricas para la validación del diseño.

MÉTODOS Y TÉCNICAS DE INVESTIGACIÓN

Para validar metodológicamente la investigación se utilizaron los siguientes métodos:

TEÓRICOS

- Histórico - Lógico: para analizar la trayectoria y evolución de la metodología de desarrollo de software y demás herramientas que se utilizan durante el trabajo.
- Método de la modelación: para el desarrollo de los artefactos es necesario la elaboración de diagramas, figuras y otros artefactos importantes, por lo que se usará el método de modelación, mediante el cual se pueden crear abstracciones con el propósito de explicar la realidad.

EMPÍRICOS

- Observación: con el objetivo de analizar y captar deficiencias.

RESULTADOS ESPERADOS

- Especificación de requerimientos de software.
- Modelo de casos de uso del sistema.
- Modelo de diseño.

ESTRUCTURA DE LA TESIS

El presente trabajo, estructurado en 3 capítulos, resume la siguiente información:

Capítulo 1. Fundamentación teórica: se realiza un estudio sobre algunas de las metodologías para el desarrollo de software, los lenguajes de modelado y herramientas que pueden utilizarse para modelar los artefactos que proponen dichas metodologías. Se tratan aspectos sobre la ingeniería de requerimientos y se exponen los principales patrones de casos de uso y diseño. Se abordan las métricas para el diseño que aseguran

la factibilidad y la calidad del modelo de casos de uso del sistema y del modelo de diseño.

Capítulo 2. Requerimientos y Diseño: se exponen las funcionalidades y restricciones del sistema mediante los requerimientos funcionales y no funcionales. Se identifican los actores del sistema. Se obtiene el diagrama de casos de uso, con la respectiva descripción de cada uno, así como los diagramas de clases del diseño y diagramas de secuencias de cada caso de uso, logrando una entrada adecuada a las actividades de implementación.

Capítulo 3. Validación de los resultados: el capítulo muestra los procedimientos y métodos utilizados para medir la calidad de los artefactos obtenidos, tanto en el modelo de casos de uso del sistema como en el de diseño.

CAPÍTULO 1

1.1 INTRODUCCIÓN

En el presente capítulo se analizan aspectos teóricos y técnicos necesarios para llevar a cabo la ingeniería de requerimiento y el diseño. Se hace alusión a algunas de las principales metodologías, herramientas y lenguajes más usados en el desarrollo de software a nivel mundial. Se muestran aspectos concernientes a la ingeniería de requerimientos, así como las diferentes etapas y técnicas para facilitar su desarrollo. Se exponen los principales patrones existentes para la concepción de los casos de uso y desarrollo del diseño. Finalmente se muestran las métricas de software que aseguran la factibilidad y la calidad del sistema.

1.2 PROCESO DE DESARROLLO DE SOFTWARE

Un sistema informático está compuesto por hardware y software. En cuanto al hardware, su producción se realiza sistemáticamente y la base de conocimiento para el desarrollo de dicha actividad está claramente definida. La fiabilidad del hardware es, en principio, equiparable a la de cualquier otra máquina construida por el hombre. Sin embargo, respecto al software, su construcción y resultados han sido históricamente cuestionados debido a los problemas asociados, entre ellos se pueden destacar los siguientes: (1)

- Los sistemas no responden a las expectativas de los usuarios.
- Los programas “fallan” con cierta frecuencia.
- Los costes del software son difíciles de prever y normalmente superan las estimaciones.
- La modificación del software es una tarea difícil y costosa.
- El software se suele presentar fuera del plazo establecido y con menos prestaciones de las consideradas inicialmente.
- Normalmente, es difícil cambiar de entorno hardware usando el mismo software.
- El aprovechamiento óptimo de los recursos (personas, herramientas, tiempo, dinero, etc.), no suele cumplirse.

Según el Centro Experimental de Ingeniería de Software (CEIS), el estudio de mercado realizado en 1996 concluyó que sólo un 16% de los proyectos de software son exitosos (terminan dentro de plazos y costos, y cumplen los requerimientos acordados).

Otro 53% sobrepasa costos y plazos, y cumple parcialmente los requerimientos. El resto ni siquiera llega al término. Algunas deficiencias comunes en el desarrollo de software son:

- Escasa o tardía validación con el cliente.
- Inadecuada gestión de los requerimientos.
- No existe medición del proceso ni registro de datos históricos.
- Estimaciones imprevistas de plazos y costos.
- Excesiva e irracional presión en los plazos.
- Escaso o deficiente control en el progreso del proceso de desarrollo.
- No se hace gestión de riesgos formalmente.
- No se realiza un proceso formal de pruebas.
- No se realizan revisiones técnicas formales e inspecciones de código.

Para lograr la productividad del software se necesita regir el proceso de desarrollo sobre metodologías que permitan obtener resultados según lo establecido con los clientes, por lo que al desarrollar un producto software uno de los aspectos más importantes a tener en cuenta es la metodología.

1.3 METODOLOGÍA DE DESARROLLO DE SOFTWARE

Las metodologías de desarrollo de software abarcan todo el ciclo de vida del software, y se definen como “un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software". Adoptando la misma se obtiene un producto software con características deseables como:

- Existencias de reglas bien definidas.
- Verificaciones en cada etapa.
- Planificación y control.
- Comunicación efectiva.
- Fácil comprensión.
- Herramientas CASE (*Computer Aided Software Engineering*).
- Soporte de la reutilización y mantenimiento de software.

De acuerdo a su definición de metodología, los procedimientos detallan consejos para elaborar una actividad; las técnicas serían la forma de ejecutar un procedimiento para obtener un resultado determinado; las herramientas software son las que hacen posible

automatizar el proceso de desarrollo del software y la documentación es la que identifica el software que se está desarrollando.

1.3.1 Rational Unified Process (RUP)

El Proceso Unificado de Desarrollo (RUP, *Rational Unified Process*) es un proceso de desarrollo de software. RUP provee un marco de trabajo genérico que puede especializarse para una gran cantidad de sistemas software. Se caracteriza por ser:

- Iterativo e incremental: cada fase se desarrolla en iteraciones. Se divide el trabajo en partes más pequeñas o mini-proyectos, donde cada uno es una iteración que resulta en un incremento.
- Centrado en la arquitectura: la arquitectura 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.
- Guiado por los 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 requerimientos. (2)

Como RUP es un proceso define “quién” está haciendo “qué”, “cuándo” y “cómo” para alcanzar un determinado objetivo. (3)

En RUP se han reunido las actividades en grupos lógicos, definiéndose nueve flujos de trabajo principales. Los seis primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. Estos flujos de trabajos son (4):

- Modelamiento del negocio: describe los procesos del negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimientos: define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y diseño: describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Pruebas: busca los defectos a lo largo del ciclo de vida del producto de software.

- Despliegue: produce liberaciones del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
- Administración de configuración y cambios: describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: control de versiones, utilización y actualización concurrente de elementos y otros.
- Administración del proyecto: involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- 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.

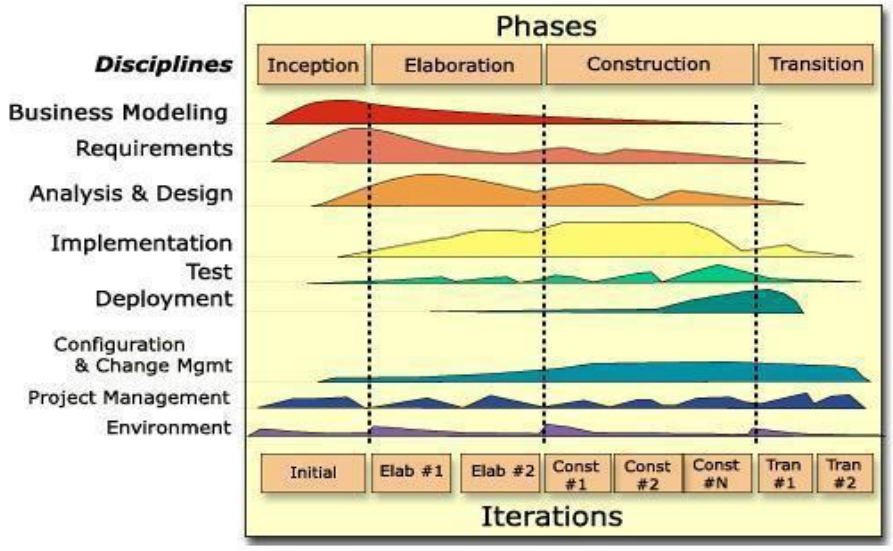


Figura 1 Fases e Iteraciones de la Metodología RUP

El ciclo de vida del software de RUP es descompuesto en cuatro fases secuenciales, cada una posee varios hitos. Al final de cada fase se realiza una evaluación para determinar que los objetivos de la fase han sido cumplidos. Una evaluación satisfactoria permite al proyecto avanzar a la próxima fase. (5)

- Inicio: el objetivo en esta etapa es determinar la visión del proyecto.
- Elaboración: en esta etapa el objetivo es determinar la arquitectura óptima.
- Construcción: en esta etapa el objetivo es llegar a obtener la capacidad operacional inicial.
- Transición: el objetivo es llegar a obtener el reléase del proyecto.

1.3.2 Extreme Programing (XP)

Programación extrema, del inglés *Extreme Programming (XP),* es una de las conocidas metodologías de desarrollo de software ágiles. La misma consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requerimientos para llegar al éxito del proyecto.

Características de XP

La metodología se basa en**:**

- Pruebas unitarias: se basan en las pruebas realizadas a los principales procesos, de tal manera que adelante lo que pudiera ocurrir en el futuro. Se hacen pruebas de las fallas que pudieran ocurrir. Es como si se adelantara a obtener los posibles errores.
- Re-fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está realizando en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

XP plantea que el análisis y diseño no se debe ausentar en un ciclo de desarrollo de software. El análisis se efectúa de forma muy ligera, mientras que el diseño (figura 2), sí se lleva a profundidad como en la mayoría de las metodologías. El análisis se transforma en la exposición por parte del cliente de las "*user-stories*", para lograr un entendimiento de lo que se quiere. Luego continúa el diseño global del sistema, en el que se profundiza hasta el nivel necesario para que los desarrolladores sepan exactamente qué van a hacer. (6)

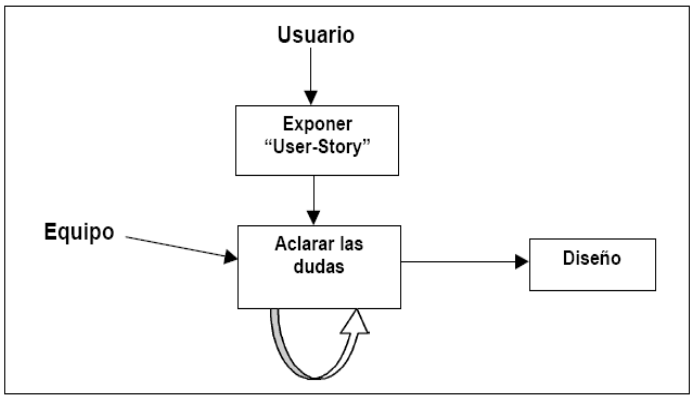


Figura 2 Actividades del diseño en XP.

1.3.3 Microsoft Solution Framework (MSF)

MSF es una metodología que se centra en los modelos de proceso y de equipo, dejando en un segundo plano las elecciones tecnológicas. Flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos. Además se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, de Proceso, de Gestión del Riesgo, de Diseño de Proceso y finalmente el de Aplicación. (7)

El proceso de desarrollo en MSF consta de 5 fases: Visión, Planeación, Desarrollo, Estabilización e Implantación; las cuales junto a los modelos mencionados anteriormente, complementan el ciclo de desarrollo de software en esta metodología. (7)

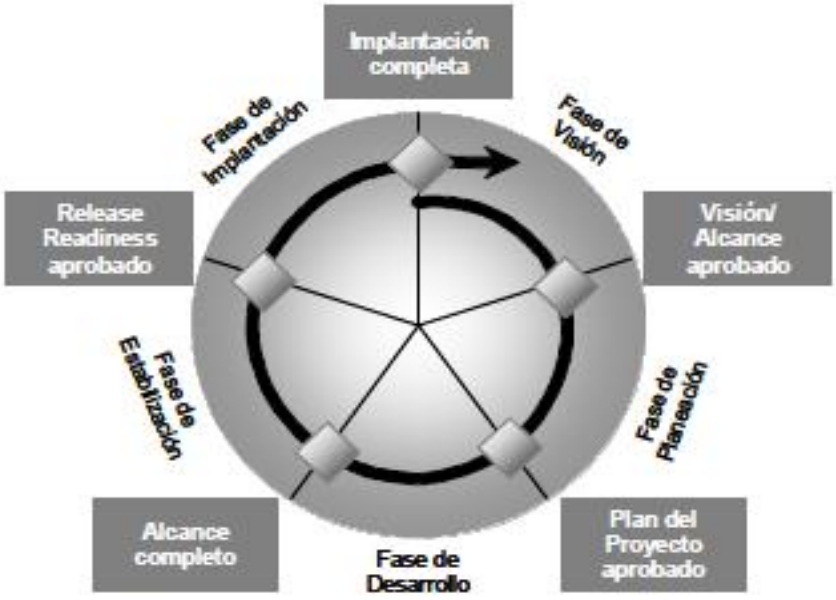


Figura 3 Modelo de Proceso de Desarrollo de Aplicaciones MSF.

- Fase 1: Visión. En esta fase el equipo y el cliente definen los requerimientos del negocio y los objetivos generales del proyecto. La fase culmina con el hito Visión y Alcance aprobados. (7)
- Fase 2: Planeación. Durante la fase de planeación el equipo crea un borrador del plan maestro del proyecto, además de un cronograma y de la especificación funcional del proyecto. Esta fase culmina con el hito Plan del proyecto aprobado. (7)
- Fase 3: Desarrollo. Esta fase involucra una serie de *releases* internos del producto, desarrollados por partes para medir su progreso y para asegurarse que

todos sus módulos o partes están sincronizados y pueden integrarse. La fase culmina con el hito Alcance completo. (7)

- Fase 4: Estabilización. Esta fase se centra en probar el producto. El proceso de prueba hace énfasis en el uso y el funcionamiento del producto en las condiciones del ambiente real. La fase culmina con el hito *Release Readiness* aprobado. (7)
- Fase 5: Implantación: En esta fase el equipo implanta la tecnología y los componentes utilizados por la solución, estabiliza la implantación, apoya el funcionamiento y la transición del proyecto, y obtiene la aprobación final del cliente. La fase termina con el hito Implantación completa. (7)

MSF enfoca el análisis y diseño mediante el Modelo de Diseño del Proceso en la fase de planeación. El mismo está diseñado para distinguir entre los objetivos empresariales y las necesidades del cliente. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible mediante un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores. (8)

Diseño conceptual: establece los conceptos que especifican las necesidades de los usuarios. Se debe comenzar a estructurar la solución propuesta, para lo cual propone la elaboración de los siguientes artefactos: (8)

- Perfil de los usuarios: especifica la ubicación, las capacidades y las expectativas de los usuarios.
- Escenarios de los usuarios: describen qué sucede en la ejecución de una tarea en particular; especifican cómo son los procesos, las funciones y los procedimientos.

Diseño lógico: estructura y organiza los elementos de la solución propuesta; para ello propone la elaboración de los siguientes artefactos: (8)

- Diseño de la Interfaz de Usuario: presenta los elementos y lineamientos que conforman el diseño de la interfaz de usuario.
- Componentes de la Solución: establece los elementos involucrados en la solución, así como sus relaciones.
- Bases de Datos Lógica: especificación lógica de las Bases de Datos que conforman o con las que interactúa la solución.

Diseño físico: en esta se aplican las restricciones tecnológicas a la solución del diseño lógico, para lo cual propone la elaboración de los siguientes artefactos: (8)

- Restricciones de Tecnología: especifica la tecnología utilizada.
- Implementación de la Interfaz del Usuario: muestra la apariencia de la solución.
- Arquitectura de la solución: presenta la vista de implantación de la solución.

1.3.4 Selección de la metodología de desarrollo de software

Luego de la realización de un estudio de las metodologías más usadas en la actualidad, se puede concluir que RUP presenta características perfectamente ajustables al proyecto y es la más conocida por el equipo de desarrollo. Esta metodología constituye uno de los estándares internacionales que más aceptación ha tenido últimamente en el desarrollo informático. Además, varias herramientas CASE soportan dicha metodología, permitiendo el trabajo en equipo y la capacidad de generar código en distintos lenguajes de programación a partir de un diseño UML.
Además, RUP es una metodología que se encarga de:

- Asegurar la producción de un software de alta calidad que reúna las necesidades de los usuarios finales en un plan y presupuesto predecibles.
- Proveer un enfoque disciplinado para asignar tareas y responsabilidades en el desarrollo del sistema.
- Reducir en gran medida el riesgo que representa la construcción de sistemas complejos, porque evoluciona de forma incremental partiendo de sistemas más pequeños en los que ya se tiene confianza.

1.4 HERRAMIENTAS CASE PARA EL DESARROLLO DE SOFTWARE

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

1.4.1 Rational Rose Enterprise Edition 2003

Rational Rose es la herramienta CASE que 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. (9)

Entre las funcionalidades que soporta el *Rational Rose* están:

- Creación de diagramas UML.
- Desarrollo orientado al modelado.
- Generación de código ADA, ANSI C ++, C++, CORBA, Java y *Visual Basic*, con capacidad de sincronización modelo-código configurables.
- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.

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

1.4.2 Enterprise Architect

Enterprise Architect es una herramienta CASE orientada a objetos que provee su alcance para el desarrollo de sistemas, administración de proyectos y análisis de negocios. Maneja totalmente el ciclo de vida de desarrollo de software, utilizando el UML como leguaje de modelado. Facilita y soporta el levantamiento de requerimientos, el análisis y diseño, las pruebas y mantenimiento del software en desarrollo. Soporta un impresionante rango de lenguajes de desarrollo, incluyendo *Action Script*, C, C++, C# y VB.NET, Java, *Visual Basic* 6, *Python*, PHP, XSD y WSDL, entre otros. Gestiona la ingeniería de código, normal e inversa, además de una efectiva documentación compatible con *Microsoft Word*. (10)

Entre las funcionalidades que soporta el *Enterprise Architect* se encuentran:

- Creación de diagramas UML.
- Creación de diagramas de Casos de uso, y Modelos lógicos, dinámicos y físicos.
- Extensiones personalizadas para modelado de procesos.

- Documentación de alta calidad compatible con *Microsoft Word*.
- Modelado de datos.
- Ingeniería directa de Base de Datos a DDL e ingeniería inversa de Base de Datos desde ODBC.
- Soporte de pruebas.
- Aplicación Multi-usuario, con sistema de seguridad.

Se integra mediante *plug-ins* con herramientas como:

- Eclipse.
- *Visual Studio.Net*

1.4.3 Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. *Visual Paradigm* ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. (11)
Entre las funcionalidades que soporta el *Visual Paradigm* se encuentran:

- Modelado de procesos del negocio.
- Modelado de base de datos.
- Generación de código.
- Ingeniería inversa.

Se integra con herramientas Java como:

- Eclipse.
- *JBuilder.*
- *NetBeans/sun ONE.*
- *Weblogic Workshop.*
- *JDeveloper.*
- *IntelliJ IDE.*

1.4.4 Selección de la herramienta CASE para el desarrollo de software

Las herramientas CASE de UML ofrecen una forma de representar sistemas complejos para facilitar una mejor comprensión mediante su código fuente y permiten desarrollar la solución correcta de software, más rápida y económicamente. *Visual Paradigm* 6.4, es la herramienta CASE seleccionada para modelar todos los artefactos del sistema. Tiene la ventaja de ser multiplataforma, soporta el ciclo de vida completo del desarrollo de software. Esta herramienta es de gran utilidad para el proyecto, por su nivel de integración con el entorno de desarrollo Java y por la posibilidad de trabajar ambos sobre la plataforma libre *GNU/Linux*, sistema operativo sobre el cual se desarrolla el proyecto.
Además, *Visual Paradigm* es una herramienta que ofrece:

- Entorno de creación de diagramas para UML.
- Diseño centrado en casos de uso y enfocado al negocio que genera 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.
- Disponibilidad de múltiples versiones.

1.5 HERRAMIENTAS DE MODELADO DE PROTOTIPOS DE INTERFAZ DE USUARIO

Los prototipos de interfaz de usuario permiten verificar los requerimientos del sistema antes de que se inicie su desarrollo. Con el diseño de los prototipos de interfaz se puede especificar el contenido, planificar las fases del desarrollo o detectar errores de diseño antes de iniciar el desarrollo del sistema.

1.5.1 Axure RP

Axure RP es una aplicación ideal para crear prototipos y especificaciones muy precisas para páginas web. Se trata de una herramienta especializada en la tarea, que cuenta con todo lo que se puede necesitar para crear los prototipos de forma más eficiente; permite componer la página web visualmente, añadiendo, quitando y modificando los elementos con suma facilidad, y donde demuestra su grado de especialización es en las anotaciones. En este punto permite especificar el estado de cada elemento (Propuesto, Aceptado, Incorporado), el beneficio esperado (Crítico, Importante, Útil), el riesgo, la estabilidad, a quién va dirigido y a quién se le asignará la tarea. (12)

Mejoras recientes en *Axure RP*:

- Fragmento de imágenes.
- Ya no hay límite de variables.
- Mejoras en la interfaz.
- Corrección de errores.

Limitaciones:

- *Axure RP* sólo funciona en los siguientes sistemas operativos: *Windows 98/98SE/Me/2000/XP*.
- Necesita *Office 2007 Compatibility Pack*.

1.5.2 Microsoft Office Visio 2007

Office Visio 2007 facilita a los profesionales empresariales la visualización, análisis y comunicación de información, sistemas y procesos complejos. Mediante los diagramas de aspecto profesional de *Office Visio 2007*, se puede mejorar la comprensión de sistemas y procesos, entender mejor la información compleja y utilizar esos conocimientos para tomar mejores decisiones a favor de la empresa. (13)

Office Visio 2007 permite documentar, diseñar y comprender de forma visual el estado de los sistemas y procesos empresariales con una gran variedad de diagramas: diagramas de flujo de proceso empresarial, diagramas de red, diagramas de flujo de trabajo, modelos de bases de datos y diagramas de software. (13)

Facilita y agiliza la creación de diagramas, tan solo con abrir una plantilla, arrastrar formas hasta los dibujos y aplicar temas para lograr un acabado perfecto. Los diagramas están organizados en plantillas según su tipo. Tiene 2 paletas desde las cuales se pueden prototipos con los objetos y controles de los formularios de Windows. La característica Autoconexión se ocupa de conectar las formas de manera automática, las distribuye de modo uniforme y las alinea con precisión. Permite además exportar a los formatos PDF y XML *Paper Specification* (XPS). (14)

1.5.3 Visual Paradigm for UML 6.4

En el epígrafe 1.4.3 se analizaron las características y funcionalidades de la herramienta *Visual Paradigm* como herramienta CASE para el desarrollo de los artefactos. Una de las funcionalidades estudiadas de *Visual Paradigm* es la que permite crear prototipos de interfaz de usuario. En un ambiente sencillo se pueden definir todos los elementos que conforman una interfaz. Permite insertar información adicional a los diagramas mediante

notas y comentarios para describir sus elementos lo que facilita la revisión de los prototipos así como el trabajo en equipo.

1.5.4 Selección de la herramienta para el modelado de Prototipos de interfaz de usuario

Después de realizar un análisis de las herramientas para el modelado de prototipos de interfaz de usuario se puede concluir que *Visual Paradigm* resuelve la necesidad de una herramienta que facilite el trabajo de diseñar interfaz de usuario en la realización de los prototipos.

1.6 LENGUAJE UNIFICADO DE MODELADO (UML)

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Es probablemente una de las innovaciones conceptuales en el mundo tecnológico del desarrollo de software que más expectativas y entusiasmos haya generado en muchos años. (15) Considerado un estándar en la industria del software creado por Grady Booch, James Rumbaugh e Ivar Jacobson.

Principales características:

1. Lenguaje unificado para la modelación de sistemas.
2. Tecnología orientada a objetos.
3. El cliente participa en todas las etapas del proyecto.
4. Permite documentar todos los artefactos de un proceso de desarrollo.
5. Corrección de errores viables en todas las etapas.
6. Aplicable para tratar asuntos de escala inherentes a sistemas complejos de misión crítica, tiempo real y cliente/servidor.

1.7 LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Un lenguaje de programación permite a uno o más programadores especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias.

1.7.1 Java

Según se indica desde *Sun Microsystems*, Java es un lenguaje creado para realizar una programación en Internet rápida y fácil, si ya se tiene conocimientos previos de C++ y de programación orientada a objetos. No se debe considerar a Java como una herramienta exclusiva y únicamente para la programación en Internet, ya que su uso puede y debe extenderse a problemas y situaciones de todo tipo; por lo tanto, para evitar confusiones, el lenguaje Java se podría definir mejor de la siguiente forma: Java es un lenguaje de programación orientado a objetos, de propósito general, que presenta características especiales que lo hacen idóneo para su uso en Internet: (16)

- Orientado a objetos: Java organiza sus programas en una colección de objetos, lo que permite estructurar los programas de una manera más eficiente y en un formato fácil de comprender. (16)
- Distribuido: Java dispone de una serie de librerías para que los programas se puedan ejecutar en varias máquinas y puedan interactuar entre sí. (16)
- Robusto: Java está diseñado para crear software altamente fiable. (16)
- Seguro: Java cuenta con ciertas políticas que evitan que se puedan codificar virus con este lenguaje, sin olvidar que existen otras restricciones que limitan lo que se puede o no hacer con los recursos críticos de una máquina. (16)
- Interpretado: la interpretación y ejecución se hace mediante la Máquina Virtual de Java (JVM) que es el entorno en el que se ejecutan los programas Java; su misión principal es la de garantizar la ejecución de las aplicaciones Java en cualquier plataforma. (16)
- Independiente de la Arquitectura: el código compilado de Java se puede usar en cualquier plataforma. (16)
- Multiejecución: Java permite elaborar programas que permitan ejecutar varios procesos al mismo tiempo sobre la misma máquina. (16)

1.7.2 Lenguajes de programación web

En la actualidad existen numerosos lenguajes de programación para desarrollar la web, tanto lenguajes de lado cliente como de lado servidor. Los de lado cliente son aquellos capaces de ser interpretados directamente por el navegador sin necesidad de un pre-tratamiento.

Entre los lenguajes de lado cliente se encuentran:

- **HTML** (acrónimo en inglés de *HyperText Markup Language*, en español Lenguaje de Marcas Hipertextuales) es un lenguaje estático para el desarrollo de sitios web, desarrollado por *World Wide Web Consortium (W3C)* (17). Los archivos pueden tener las extensiones html. Entre sus ventajas están que es admitido por todos los exploradores, sencillo y de archivos pequeños. Como desventaja resalta que la interpretación de cada navegador puede ser diferente.
- ***JavaScript*** es un lenguaje interpretado creado por Brendan Eich en la empresa *Netscape Communications*. El código *JavaScript* puede ser integrado en las páginas web. El W3C para evitar incompatibilidades diseñó un estándar para la modelación de objetos del documento, más conocido como DOM (en inglés *Document Object Model*). Entre sus ventajas está su *scripting* seguro y fiable, y contradictoriamente su debilidad es su visibilidad al alcance de cualquier usuario.

Los lenguajes de lado servidor son aquellos reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible.
Entre los lenguajes de lado servidor se encuentran:

- **ASP.NET** es un lenguaje comercializado por *Microsoft*. Fue creado para desarrollar tanto web sencillas o grandes aplicaciones web y resolver las limitantes de su antecesor ASP. Entre sus ventajas están que es un lenguaje completamente orientado a objetos, su velocidad de respuesta del servidor y su seguridad. El consumo de recursos es su desventaja. Para el desarrollo en ASP.NET se puede utilizar C#, VB.NET o J#. Su funcionamiento requiere tener instalado IIS (*Internet Information Server*) con el *Framework .Net*.
- **Java JSP** (*Java Server Pages*), desarrollado por *Sun Microsystems*, está orientado a desarrollar páginas web en Java. Comparte ventajas similares a la de ASP.NET y fue desarrollado principalmente para la creación de aplicaciones web potentes. Posee un motor de páginas basado en los *servlets* de Java. Necesita tener instalado un servidor *Tomcat* para su funcionamiento. Comparado por su potencialidad con ASP.NET, destaca por ser multiplataforma. Resulta un tanto complejo en su aprendizaje.
- **PHP** es un lenguaje de *script* interpretado utilizado para la generación de páginas web dinámicas, embebidas en páginas HTML y ejecutadas en el servidor. Para su funcionamiento necesita tener instalado Apache o IIS con las librerías de PHP. La mayor parte de su sintaxis ha sido tomada de C, Java y *Perl* con algunas

características específicas. Es un lenguaje multiplataforma, fácil de aprender y no requiere definición de tipos de variables, siendo estas sus mayores ventajas. Entre sus desventajas se encuentra la pobre programación orientada a objetos para aplicaciones grandes, además de dificultar la organización por capas de la aplicación.

La integración entre lenguajes de lado servidor y de lado cliente hacen posible la implementación y con ello la solución a muchos de los problemas actuales.

1.7.3 Selección del lenguaje de programación que será utilizados en la solución

Después de analizar varios lenguajes de programación, teniendo en cuenta las necesidades del cliente y el tipo de aplicación a realizar, se decidió utilizar Java para realizar la implementación de la solución, por las siguientes características:

- Multiplataforma
- Orientado a objetos
- Distribuido
- Robusto
- Dinámico
- Seguro

1.8 INGENIERÍA DE REQUERIMIENTOS

Para lograr un mejor entendimiento de lo que es la Ingeniería de Requerimientos (IR) y lo que representa, es necesario primero comprender qué son los requerimientos, cómo están definidos y para qué sirven. Estas son algunas de sus definiciones:

- 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. (18)
- Una condición o capacidad necesaria dada por un usuario con el objetivo de resolver un problema o alcanzar un objetivo. (19)
- Expresan las necesidades y restricciones atribuibles a un producto de software que contribuye a la solución de algún problema del mundo real. (20)

Los requerimientos pueden ser clasificados como funcionales o no funcionales. Los funcionales describen qué es lo que el sistema debe hacer para dar soporte a las funciones y objetivos del usuario, y deben responder a las siguientes preguntas: (20)

- ¿Cómo las entradas son transformadas en salidas?
- ¿Quién inicia y recibe información específica?
- ¿Qué información debe estar disponible para que cada función sea ejecutada?

Los requerimientos no funcionales imponen restricciones de cómo los requerimientos funcionales deben ser implementados. (20) Una vez definidos los requerimientos se puede comprender mejor qué es y qué abarca la IR. Para responder a las interrogantes formuladas, hay que remitirse a las definiciones de los autores e instituciones más citados de la IR:

- La Ingeniería de Requerimientos del software es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refina en detalle los requerimientos del sistema y el papel asignado al software. Se crean modelos de los requerimientos de datos, flujos de información y control, y del comportamiento operativo. Se analizan alternativas y el modelo completo del análisis es creado. (21)
- La Ingeniería de Requerimientos se ocupa de la elicitación, análisis, especificación y validación de requerimientos de software. (22)
- Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. (19)
- Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal. (19)

1.8.1 Etapas fundamentales

La IR se divide en etapas bien definidas, que especifican pasos, tareas y técnicas que se deben emplear para interactuar con los clientes y especificar correctamente el sistema. El buen entendimiento de cada etapa de la IR y su correcta aplicación durante la primera fase del desarrollo del software, propicia una vía efectiva para mantener una fluida comunicación entre los involucrados en el proyecto y una guía para especificar en detalles el sistema solicitado.

Elicitación de requerimientos

La captura de requerimientos es la actividad mediante la cual el equipo de desarrollo de un sistema de software extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema. (23) Es fundamentalmente una actividad humana, donde los *stakeholders* se identifican y comienzan a establecerse las relaciones entre el equipo desarrollador y el cliente. También se le conoce como “Captura de requerimientos”, “Descubrimiento de requerimientos” y “Adquisición de requerimientos”.

En esta etapa se acude a varias fuentes de información. Algunos autores consideran que la principal fuente de información son los usuarios de una organización. Otros utilizan en cambio a los expertos del dominio, de los cuales obtiene información de forma oral. (24)

Análisis de requerimientos

Esta es la etapa de derivación de requerimientos del sistema de software a través de la observación de sistemas existentes, discusión con usuarios y proveedores de información potenciales, análisis de tareas y así sucesivamente. Esto puede involucrar el desarrollo de uno o más modelos diferentes del sistema, que ayudan al analista a entender el sistema que será especificado. También pueden desarrollarse prototipos para ayudar a entender los requerimientos. (25)

En esta etapa se trata, precisamente, de analizar la información recibida desde los usuarios, para distinguir las necesidades de tareas, los requerimientos funcionales, atributos de calidad y soluciones sugeridas. (26) Se clasifican los requerimientos identificados y se eliminan todas las inconsistencias y ambigüedades que los mismos posean.

Se transforman los requerimientos informales en requerimientos técnicos mediante el aseguramiento de que los mismos reflejen los atributos de calidad y que expresen las necesidades de los clientes. El análisis es una actividad iterativa. Los pasos del proceso deberán ser repetidos una cierta cantidad de veces. (22)

Especificación de requerimientos

En esta actividad se detalla el documento Especificación de Requerimientos de Software (ERS). Conjuntamente se puede realizar un diseño de alto nivel para ayudar a descubrir errores en la definición de los requerimientos, los cuales deben ser corregidos.

Las especificaciones forman un puente entre requerimientos y diseño. Sus entradas son el documento de requerimientos y la información de los usuarios y desarrolladores. La especificación de un sistema de software describe aspectos funcionales y de calidad. (25)

Los clientes y usuarios utilizan la Especificación de Requerimientos para comparar si lo que se está proponiendo coincide con las necesidades de la empresa. Los analistas y programadores la utilizan para determinar el producto que debe desarrollarse. El personal

de pruebas elaborará las pruebas funcionales y de sistemas en base a este documento. Para el administrador del proyecto sirve como referencia y control de la evolución del sistema. (26)

Las estrategias recomendadas para la especificación de los requerimientos del software están descritas por IEEE 830-1998. Este estándar describe las estructuras posibles, contenido deseable, y calidades de una especificación de requerimientos del software. (27)

Verificación y validación de requerimientos

Muchos de los autores utilizan a veces los términos verificación y validación como sinónimos. Generalmente la verificación y validación suelen asociarse a los tipos de técnicas de prueba, principalmente sobre el producto final.

Existen varios estándares como el 1012 de la IEEE de 1997 (*Standard for Software Verification and Validation Plans*) y el 1059 (*Guide for Software Verification and Valitation Plans*), que considera la verificación y validación como actividades casi inseparables. En el glosario de la IEEE de 1990 aparecen ambos términos en una única entrada: (19)

Verificación y validación (1): *el proceso de determinar si los requerimientos para un sistema o componente son completos y correctos, los productos de cada fase de desarrollo satisfacen los requerimientos o condiciones impuestas por la fase previa y el sistema o componente final es acorde con los requerimientos especificados.*

Verificación (2): *(**a**) el proceso de evaluar un sistema o componente para determinar si los productos de una fase de desarrollo dada satisfacen las condiciones impuestas al comienzo de la fase. (**b**) prueba formal de la corrección de un programa.*

Validación (2): *el proceso de evaluar un sistema o componente durante o al final del proceso de desarrollo para determinar si satisface los requerimientos especificados.*

Otras definiciones menos formales se pueden encontrar de la siguiente forma:

Verificación (3): *¿se está construyendo correctamente el producto?*

Validación (3): *¿se está construyendo el producto correcto?*

En este trabajo se tratan los términos atribuyéndoles significados distintos, especialmente en la Ingeniería de Requerimientos, que no tienen que ir necesariamente unidos. Por verificación se entenderá el conjunto de actividades relacionadas con la calidad de las especificaciones de requerimientos respecto a un conjunto de propiedades deseables. Por validación de requerimientos, el conjunto de actividades encaminadas a llegar a un

acuerdo entre todos los participantes, en el que se ratifique que los requerimientos elicitados y analizados representan realmente las necesidades de clientes y usuarios.

Administración de requerimientos

La administración de requerimientos es un conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar y seguir los requerimientos y los cambios en cualquier momento. (26)

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

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

La administración de los requerimientos incluye todas las actividades que mantienen la integridad y exactitud de los requerimientos a medida que el proyecto progresa. En esta etapa se enfatizan: (21)

- Control de los cambios de los requerimientos que están sobre la línea base definida.
- Control de versiones tanto de requerimientos individuales como del documento de requerimientos.

1.8.2 Técnicas utilizadas en la Ingeniería de Requerimientos

La ingeniería de requerimientos puede ser un proceso largo y arduo. Para minimizar sus dificultades se aplican técnicas de obtención de la información, como las siguientes:

Entrevistas

La entrevista consiste en el intento sistemático de recoger información de otra persona a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada. Cada tipo de entrevista requiere un comportamiento y una preparación distinta: selección, disciplina y rendimiento.

Cualidades del Entrevistador:

- Saber observar y escuchar
- Poseer madurez

- Ser objetivo e imparcial
- No ser autoritario
- Capacidad de “empatía”
- Comprensión
- Ser cordial y accesible
- Respetar la intimidad
- Ser sincero, paciente, sereno
- Ser prudente

Cuestionarios

El cuestionario consiste en preguntas sobre varios aspectos de un sistema. Por lo común, los encuestados son usuarios de los sistemas existentes o usuarios en potencia del sistema propuesto. En algunos casos, son gerentes o empleados que proporcionan datos para el sistema propuesto o que serán afectados por él.

Con frecuencia se utilizan preguntas abiertas para descubrir sentimientos, opiniones y experiencias generales, o para explorar un proceso o problema. (25)

Las preguntas pueden ser enfocadas a un elemento del sistema, tales como usuarios, procesos, etc. El siguiente ejemplo muestra algunos tipos de preguntas abiertas.

- ¿Quién es el cliente?
- ¿Quién es el usuario?
- ¿Son sus necesidades diferentes?
- ¿Cuáles son sus habilidades, capacidades, ambiente?
- ¿Cuál es la razón por la que se quiere resolver este problema?
- ¿Cuál es el valor de una solución exitosa?
- ¿Cómo usted resuelve el problema actualmente?
- ¿Qué retrasos ocurren o pueden ocurrir?
- ¿Qué problemas podría causar este producto en el negocio?
- ¿En qué ambiente se usará el producto?
- ¿Cuáles son sus expectativas para los conceptos fácil de usar, confiable, rendimiento?

Tormenta de ideas

La tormenta de ideas es una técnica simple para la generación de ideas. Se reúne un conjunto de personas relacionadas al producto y éstas sugieren y exploran sus ideas libremente. Generalmente las reuniones se realizan con 4 a 10 personas; una de ellas deberá actuar como líder para comenzar pero no para restringir la expresión de los restantes participantes. Se genera una amplia variedad de puntos de vistas, se estimula el pensamiento creativo, se logra construir una imagen más completa del problema, se provee un ambiente social mucho más confortable y de fácil aprendizaje. (25)

Sistemas existentes

Esta técnica consiste en analizar distintos sistemas ya desarrollados que estén relacionados con el sistema a ser construido. Por un lado, es posible analizar las interfaces de usuario, observando el tipo de información que se maneja y cómo es manejada, por otro lado también es útil analizar las distintas salidas que los sistemas producen (listados, consultas, etc.) porque siempre pueden surgir nuevas ideas sobre la base de estas. (22)

Prototipos

Un prototipo es una pequeña muestra, de funcionalidad limitada, de cómo sería el producto final una vez terminado. Ayuda a conocer la opinión de los usuarios y rectificar algunos aspectos antes de llegar al producto terminado.

Casos de uso

Los casos de uso son una técnica para especificar el comportamiento de un sistema. Los casos de uso permiten entonces describir la posible secuencia de interacciones entre el sistema y uno o más actores, en respuesta a un estímulo inicial proveniente de un actor, es una descripción de un conjunto de escenarios, cada uno de ellos comenzado con un evento inicial desde un actor hacia el sistema. La mayoría de los requerimientos funcionales se pueden expresar con casos de uso. Según el autor Sommerville, los casos de uso son una técnica que se basa en escenarios para la obtención de requerimientos. (21)

1.9 DISEÑO DEL SOFTWARE

1.9.1 Principios del diseño

El diseño de software es una representación significativa de ingeniería de algo que se va a construir, basándose en los requerimientos del cliente. Al mismo tiempo la calidad se puede evaluar y comparar con los criterios predefinidos para obtener un diseño bueno. En el contexto de la ingeniería del software, el diseño se centra en cuatro áreas

importantes: datos, arquitectura, interfaces y componentes. En estas cuatro áreas se aplican los principios que se abordan a continuación. (28)

- En el proceso de diseño no deberá utilizarse “orejeras”. Un buen diseñador deberá tener en cuenta enfoques alternativos, juzgando todos los que se basan en los requerimientos del problema, los recursos disponibles para realizar el trabajo y los conceptos de diseño. (28)
- El diseño deberá poderse rastrear hasta el modelo de análisis. Dado que un solo elemento del modelo de diseño suele hacer un seguimiento de los múltiples requerimientos, es necesario tener un medio de rastrear como se han cumplido los requerimientos por el modelo de diseño. (28)
- Los sistemas se construyen utilizando un conjunto de patrones de diseño, muchos de los cuales probablemente ya se han encontrado antes. El tiempo de diseño se deberá invertir en la representación verdadera de ideas nuevas y en la integración de esos patrones que ya existen. (28)
- El diseño deberá minimizar la distancia intelectual entre el software y el problema como si de la misma vida real se tratara. Es decir, la estructura del diseño del software (siempre que sea posible) imita la estructura del dominio del problema. (28)
- El diseño deberá presentar uniformidad e integración. Un diseño es uniforme si parece que fue una persona la que lo desarrolló por completo. Las reglas de estilo y de formato deberán definirse para un equipo de diseño antes de comenzar el trabajo sobre el diseño. Un diseño se integra si se tiene cuidado a la hora de definir interfaces entre los componentes del diseño. (28)
- El diseño deberá estructurarse para admitir cambios. Los conceptos de diseño estudiados hacen posible un diseño que logra este principio. El diseño deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operación aberrantes. Un software bien diseñado no deberá nunca explotar, deberá diseñarse para adaptarse a circunstancias inusuales y si debe terminar de funcionar, que lo haga de forma suave. (28)
- El diseño no es escribir código y escribir código no es diseñar. Incluso cuando se crean diseños procedimentales para componentes de programas, el nivel de abstracción del modelo de diseño es mayor que el código fuente. Las únicas decisiones de diseño realizadas a nivel de codificación se enfrentan con pequeños datos de implementación que posibilitan codificar el diseño procedimental. (28)

- El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminarlo. Para ayudar al diseñador en la evaluación de la calidad se dispone de conceptos de diseño y de medidas de diseño. (28)
- El diseño deberá revisarse para minimizar los errores conceptuales. A veces existe la tendencia de centrarse en minucias cuando se revisa el diseño, olvidándose del bosque por culpa de los árboles. Un equipo de diseñadores deberá asegurarse de haber afrontado los elementos conceptuales principales antes de preocuparse por la sintaxis del modelo del diseño. (28)

¿Quién lo hace? El ingeniero del software es quien diseña los sistemas basados en computadoras, pero los conocimientos que se requieren en cada nivel de diseño funcionan de diferentes maneras. En el nivel de datos y de arquitectura, el diseño se centra en los patrones de la misma manera a como se aplican en la aplicación que se va a construir. En el nivel de la interfaz, es la ergonómica humana la que dicta nuestro enfoque de diseño. Y en el nivel de componentes, un "enfoque de programación" conduce a diseños de datos y procedimentales eficaces. (28)

¿Por qué es importante? Si se construye una casa, ¿se hace sin un plano? Se correrían riesgos, se cometerían errores, habría un plano de casa sin sentido, con ventanas y puertas en sitios equivocados…un desastre. El software de computadora es considerablemente más complejo que una casa, de aquí que se necesita un plano "el diseño". (28)

¿Cuáles son los pasos? El diseño comienza con el modelo de los requerimientos. Se trabaja por transformar este modelo y obtener cuatro niveles de detalles de diseño: la estructura de datos, la arquitectura del sistema, la representación de la interfaz y los detalles a nivel de componentes. Durante cada una de las actividades del diseño, se aplican los conceptos y principios básicos que llevan a obtener una alta calidad. (28)

¿Cuál es el producto obtenido? Por último se produce una especificación del diseño. La especificación se compone de los modelo del diseño que describen los datos, arquitectura, interfaces y componentes. Cada una de esta parte es la que forma el producto obtenido del proceso de diseño. (28)

¿Cómo puedo estar seguro de que lo he hecho correctamente? En cada etapa de revisión los productos del diseño del software en cuanto a claridad, corrección, finalización y consistencia, y se comparan con los requerimientos y unos con otros. (28)

1.10 PATRONES DE CASOS DE USO Y PATRONES DE DISEÑO

1.10.1 Patrones de casos de uso

La experiencia en la utilización de casos de uso en los proyectos de los sistemas más diversos ha evolucionado en un conjunto de patrones a través de los años. Estas técnicas han demostrado ser la solución preferida por la comunidad del desarrollo de software. Se presentan como herramientas que permiten resolver los problemas de una forma ágil y sistemática. (29)

Concordancia: Reutilización (Commonality: Reuse)

El patrón Concordancia: Re-uso es un patrón de estructura que consiste en tres casos de uso. El primero llamado “Sub-secuencia Común”, modela la secuencia de acciones que aparecen en múltiples casos de uso del modelo. Los otros dos casos de uso comparten de esta sub-secuencia común de acciones (dos es la menor cantidad que puede existir).

La sub-secuencia tiene que estar en un fragmento, es decir, todo lo que requiere estar incluido tiene que estar en un único fragmento completo. Además no se puede hacer referencia desde la sub-secuencia a donde esta es utilizada, porque el caso de uso incluido tiene que ser independiente del caso de uso base. (29)

Concordancia: Especialización (Commonality: Specialization)

Otro patrón de concordancia que contiene casos de uso del mismo tipo. En este caso, ellos son modelados como especializaciones de un caso de uso de uso común. Todas las acciones del caso de uso común son heredadas por sus casos de uso hijos, en donde otras acciones pueden ser agregadas o las acciones heredadas pueden ser especializadas. Este patrón es aplicable cuando las acciones modeladas por los casos de uso son del mismo tipo, y este tipo debería ser hecho visible en el modelo. (29)

Extensión o Inclusión Concreta: Extensión (Concrete Extension or Inclusion: Extension)

El patrón Extensión o Inclusión Concreta: Extensión consiste en dos casos de uso y una relación de extensión entre ellos. El caso de uso extendido es concreto; es decir, puede ser instanciado por el mismo, así como también puede extender del caso de uso base.

El patrón es aplicable cuando un flujo puede extender el flujo de otro caso de uso, así como también puede ser realizado por el mismo. (29)

Extensión o Inclusión Concreta: Inclusión (Concrete Extension or Inclusion: Inclusion)

En este patrón, existe una relación de inclusión desde el caso de uso base al caso de uso incluido. El último puede ser instanciado por el mismo. El caso de uso base podría ser cualquiera, concreto o abstracto. El patrón es usado cuando un flujo podría ser incluido en el flujo de otro caso de uso y además puede ser realizado por el mismo. (29)

CRUD: Completo

El patrón CRUD: Completo consiste en un caso de uso, llamado Gestionar Información, modelando todos las diferentes operaciones que pueden ser realizadas en una pieza de información de cierto tipo, como crearla, leerla, actualizarla y eliminarla. Este patrón debería ser usado cuando todos los flujos contribuyen al mismo valor de negocio y son cortos y sencillos. (29)

Actores Múltiples: Roles distintos (Multiple Actors: Distinct Roles)

Consiste en un caso de uso y al menos dos actores. Es usado cuando dichos actores juegan diferentes roles en relación con el caso de uso, es decir, ellos interactúan diferentemente con el caso de uso. (29)

Actores Múltiples: Roles comunes (Multiple Actors: Common Roles)

Es un patrón alternativo, los dos actores juegan el mismo rol en relación con el caso de uso. Este rol es representado por otro actor, del cual heredan los dos actores anteriores que comparten el mismo rol. El patrón es aplicable cuando, desde el punto de vista del caso de uso, solo existe una entidad externa interactuando con cada instancia del caso de uso. (29)

1.10.2 Patrones de diseño

Los patrones de diseño (*design patterns*) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. (30)
Los patrones se clasifican según su propósito en:

Patrones de Creación: Tratan la creación de instancias o sobre qué objetos un objeto delegará responsabilidades. (30)

- *Abstract Factory*: Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta.
- *Builder*: Permite a un objeto construir un objeto complejo especificando sólo su tipo y contenido.
- *Factory Method*: Define una interfaz para crear un objeto dejando a las subclases decidir el tipo específico al que pertenecen.

- *Prototype*: Permite a un objeto crear objetos personalizados sin conocer su clase exacta a los detalles de cómo crearlos.
- *Singleton*: Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él.

Patrones Estructurales: Tratan la relación entre clases, la combinación de clases y la formación de estructuras de mayor complejidad, describiendo así la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros. (30)

- *Adapter*: Convierte la interfaz que ofrece una clase en otra esperada por los clientes.
- *Bridge*: Desacopla una abstracción de su implementación y les permite variar independientemente.
- *Composite*: Permite gestionar objetos complejos e individuales de forma uniforme.
- *Decorator*: Extiende la funcionalidad de un objeto dinamizante de tal modo que es transparente a sus clientes.
- *Facade*: Simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto de comunicación.
- *Flyweight*: Usa la compartición para dar soporte a un gran número de objetos de grano fino de forma eficiente.
- *Proxy*: Proporciona un objeto con el que se controla el acceso a otro objeto.

Patrones de Comportamiento: Tratan la interacción y cooperación entre clases. Organizan, manejan y combinan comportamientos. (30)

- *Chain of Responsability*: Evita el acoplamiento entre quien envía una petición y el receptor de la misma.
- *Command*: Encapsula una petición de un comando como un objeto.
- *Interpreter*: Dado un lenguaje define una representación para su gramática y permite interpretar sus sentencias.
- *Iterator*: Acceso secuencial a los elementos de una colección.
- *Mediator*: Define una comunicación simplificada entre clases.
- Memento: Captura y restaura un estado interno de un objeto.
- *Observer*: Una forma de notificar cambios a diferentes clases dependientes.
- *State*: Modifica el comportamiento de un objeto cuando su estado interno cambia.

- *Strategy*: Define una familia de algoritmos, encapsula cada uno y los hace intercambiables.
- *Template Method*: define un esqueleto de algoritmo y delega partes concretas de un algoritmo a las subclases.
- *Visitor*: Representa una operación que será realizada sobre los elementos de una estructura de objetos, permitiendo definir nuevas operaciones sin cambiar las clases de los elementos sobre los que opera.

1.11 MÉTRICAS PARA VALIDAR EL DISEÑO

Según aparece definido en el Glosario Estándar de Términos de la Ingeniería de Software de la IEEE, la métrica es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

La medida es un factor clave en el desarrollo del software ya que: ayuda a entender qué está pasando durante el desarrollo y el mantenimiento, permite controlar el desarrollo del proyecto, y estimula la mejora de procesos y productos. (31)

1.11.1 Métricas de diseño arquitectónico

Tienen en cuenta las características relativas a la estructura y eficiencia de los componentes que forman la arquitectura de un sistema.

La complejidad estructural $S(i)$ de un módulo i se determina usando la ecuación

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

donde $f_{out}\ (i)$ es la expansión del módulo.

La complejidad de datos $D(i)$ de un módulo i se define como la proporción entre el número de variables de entrada y salida del módulo $V(i)$ y su expansión $f_{out}\ (i)$.

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

La complejidad del sistema $C(i)$ se calcula sumando las complejidades estructural y de datos.

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

A medida que disminuyen los valores de complejidad del sistema, su complejidad arquitectónica disminuye también.

1.11.2 Métricas de diseño a nivel de componente

Se basan en las características internas de los componentes de software permitiendo valorar la calidad del diseño en cuanto a cohesión, acoplamiento y complejidad. (21)

Cohesión Funcional Fuerte

La siguiente fórmula es empleada para determinar la cohesión funcional fuerte. En la misma $SU(SA(i))$ representa el número de señales de super-unión (el conjunto de señales de datos que se encuentran en todas las porciones de datos de un módulo i):

$$CFF(i) = \frac{SU(SA(i))}{señales(i)}$$

Entre más cercano estén los valores de CFF a 1 mayor será la cohesión del módulo. (21)

1.11.3 Métricas orientadas a clases

Tamaño de clase (TC)

El tamaño general de una clase se puede determinar empleando las medidas siguientes:

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

Si existen valores grandes de TC estos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilización de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente. (21)

Árbol de Profundidad de Herencia (APH)

Esta métrica se define como la longitud máxima desde el nodo hasta la raíz del árbol de herencia. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de PH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de PH implican que se pueden reutilizar muchos métodos. (21)

1.12 CONCLUSIONES

En este capítulo se definieron las metodologías y herramientas a utilizar en el desarrollo de la solución y se estipularon las métricas correspondientes. Se arribó a las siguientes conclusiones:

- Emplear RUP como metodología de desarrollo, por ser una guía poderosa para el desarrollo del sistema.
- Utilizar Visual Paradigm como herramienta CASE de desarrollo, que permitirá el modelado de los artefactos del módulo Fichas Planes de Personal del sistema SINAPSIS.
- Utilizar Visual Paradigm for UML 6.4 como herramienta para la creación de los prototipos interfaz de usuario.
- Utilizar el lenguaje de programación Java, por ser una plataforma empresarial libre que permitirá un desarrollo robusto y rápido del sistema.
- Aplicar las etapas de identificación, análisis, especificación y validación de los requerimientos, aplicando cada una de las actividades que estas proponen en la ingeniería de requerimientos.
- Aplicar las siguientes técnicas de ingeniería de requerimientos: entrevistas, tormenta de ideas y sistemas existentes para obtener la información de los clientes.
- Aplicar patrones de casos de uso, que permitirá la identificación y elaboración del diagrama de casos de uso del módulo Fichas Planes de Personal del sistema SINAPSIS.
- Aplicar los patrones de diseño necesarios para no cometer errores tradicionales en el diseño del módulo Fichas Planes de Personal del sistema SINAPSIS.
- Utilizar métricas para el chequeo y validación de los artefactos del módulo Fichas Planes de Personal del sistema SINAPSIS.

CAPÍTULO 2

2.1 INTRODUCCIÓN

En el presente capítulo se describe la solución que se propone para la realización de la ingeniería de requerimientos y el diseño del módulo Fichas Planes de Personal del sistema SINAPSIS. Se exponen las funcionalidades y restricciones del sistema mediante los requerimientos funcionales y no funcionales. Luego se muestra el análisis del sistema correspondiente a dichos requerimientos, mediante el modelo de casos de uso del sistema; y se analizan los casos de uso y sus especificaciones para modelar los artefactos imprescindibles para obtener un adecuado modelo de diseño.

Modelo de casos de uso del sistema

Este modelo permite que los desarrolladores del software y los clientes lleguen a un acuerdo sobre las condiciones y posibilidades que este debe cumplir. Entre los principales objetivos que tiene el modelo está identificar los requerimientos funcionales y no funcionales. Luego se definen los actores, casos de uso y la descripción detallada de cada uno de ellos, que junto al diagrama de casos de uso del sistema, permitirán una posterior modelación del diseño.

Modelo de diseño

Este modelo describe la realización física de los casos de uso centrándose en cómo los requerimientos funcionales y no funcionales tienen impacto en el sistema a considerar. Es utilizada como entrada fundamental de las actividades de implementación. Para su desarrollo se identificarán subsistemas y paquetes de diseño, así como diagramas de clases del diseño, de secuencia, de clases persistentes y modelo de datos.

2.2 REQUERIMIENTOS DE SOFTWARE

Los requerimientos se obtienen de recopilar, analizar y verificar las necesidades del cliente para un sistema. Estos son analizados y son agrupados en funcionales y no funcionales. Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir; no alteran la funcionalidad del producto y se mantienen invariables sin importarle con qué propiedades o cualidades se relacionen. Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Se clasifican según las categorías de Usabilidad, Fiabilidad, Eficiencia, Seguridad, Portabilidad, Reusabilidad y Capacidad. (20)

2.2.1 Requerimientos funcionales

RF_FP.1 Registrar datos básicos

El sistema permitirá registrar los datos básicos de una ficha de planes de personal. Los campos que conforman los datos básicos son los siguientes:

- Entidad de adscripción
- Entidad responsable
- Objetivo Estratégico
- Diagnóstico funcional
- Objetivo o Meta funcional

RF_FP.2 Modificar datos básicos

El sistema permitirá modificar los datos básicos de una ficha de planes de personal. Los campos modificables son los siguientes:

- Entidad de adscripción
- Entidad responsable
- Objetivo Estratégico
- Diagnóstico funcional
- Objetivo o Meta funcional

RF_FP.3 Asignar tipo de RRHH

El sistema permitirá asignar un nuevo tipo de Recurso Humano, especificándose además el número de personal y su correspondiente Estructura de Costo Anual (Costo Directo). Los campos necesarios son los siguientes:

- Subsistema de Recursos Humanos
- Acciones
- Recurso Humano
- Acción Centralizada
    - Cantidad Femenino
    - Cantidad Masculino
- Proyecto
    - Cantidad Femenino
    - Cantidad Masculino
- Estructura de Costo Anual (Costo Directo)
    - Cantidad por Acción Centralizada
    - Cantidad por Proyecto

RF_FP.4 Modificar asignación de tipo de RRHH

El sistema permitirá modificar una asignación de tipo de Recurso Humano que se haya hecho anteriormente. Los campos necesarios son los siguientes:

- Subsistema de Recursos Humanos
- Acciones
- Recurso Humano
- Acción Centralizada
    - Cantidad Femenino
    - Cantidad Masculino
- Proyecto
    - Cantidad Femenino
    - Cantidad Masculino
- Estructura de Costo Anual (Costo Directo)
    - Cantidad por Acción Centralizada
    - Cantidad por Proyecto

RF_FP.5 Eliminar asignación de tipo de RRHH

El sistema permitirá eliminar una asignación de tipo de Recurso Humano que se haya hecho anteriormente.

RF_FP.6 Mostrar listado de tipo de Recursos Humanos asignados

El sistema permitirá mostrar el listado de todos los tipos de Recursos Humanos asignados a una Acción perteneciente a un Subsistema de Recursos Humanos. Los campos que se muestran son los siguientes:

- Recurso Humano
- Acción Centralizada
    - Cantidad Femenino
    - Cantidad Masculino
    - Total
- Proyecto
    - Cantidad Femenino
    - Cantidad Masculino
    - Total
- Estructura de Costo Anual (Costo Directo)
    - Cantidad por Acción Centralizada
    - Cantidad por Proyecto
    - Subtotal

Para consultar todos los requerimientos funcionales, ver **Anexo 1**.

2.2.2 Requerimientos no funcionales

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.8 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.9 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. Según el nivel de concurrencia, debe ser capaz de prestar servicio sin que se deterioren los tiempos de respuestas.

Seguridad

RNF.10 Permitir el intercambio de datos entre el cliente y el servidor por canales cifrados

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

Portabilidad

RNF.12 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.15 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.17 Considerar características técnicas mínimas para la ejecución en clientes

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

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

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

Para consultar los todos los requerimientos no funcionales, ver **Anexo 2**.

2.3 ACTORES DEL SISTEMA

Un actor no es más que un conjunto de roles que los usuarios desempeñan cuando interaccionan con los casos de uso. Los actores representan a terceros fuera del sistema que colaboran con el mismo. Una vez que se identificaron los actores del sistema, se tiene identificado el entorno externo del sistema. (21)

Los actores del sistema que interactúan con las funcionalidades del módulo Ficha Planes de Personal son:

Actor	Descripción
Planificador Ficha Planes	El actor “Planificador Ficha Planes” es un usuario del sistema que puede crear una ficha planes de personal o modificar una existente. Es el encargado además de asignar recursos humanos a la ficha planes de personal, así como de la asignación de las incidencias indirectas. Este actor se encarga también de actualizar la programación física-presupuestaria, ver los detalles de la Estructura de Costo Anual y, además, generar reporte de Incidencias indirectas y reporte general de Ficha Planes.
Administrador central	El actor **“**Administrador central” es un usuario del sistema que tiene permisos para gestionar (crear, modificar y eliminar) los objetos que pueden cambiar con el paso del tiempo (subsistemas, acciones, incidencias indirectas y otros que se mencionan a continuación de algunos casos de usos descritos).
Usuario	El actor “Usuario” puede acceder a los reportes de Incidencias Indirectas y al reporte general de ficha planes que le permita ver de manera general el comportamiento de la ficha enfocado a distintos aspectos como las incidencias indirectas, subsistemas y acciones, entre otros.

2.4 DIAGRAMA DE CASOS DE USO DEL SISTEMA

Luego de identificados los requerimientos, los actores y casos de usos del sistema, se estructura el diagrama de casos de usos del sistema, que representa la relación de los actores del sistema con los casos de uso.

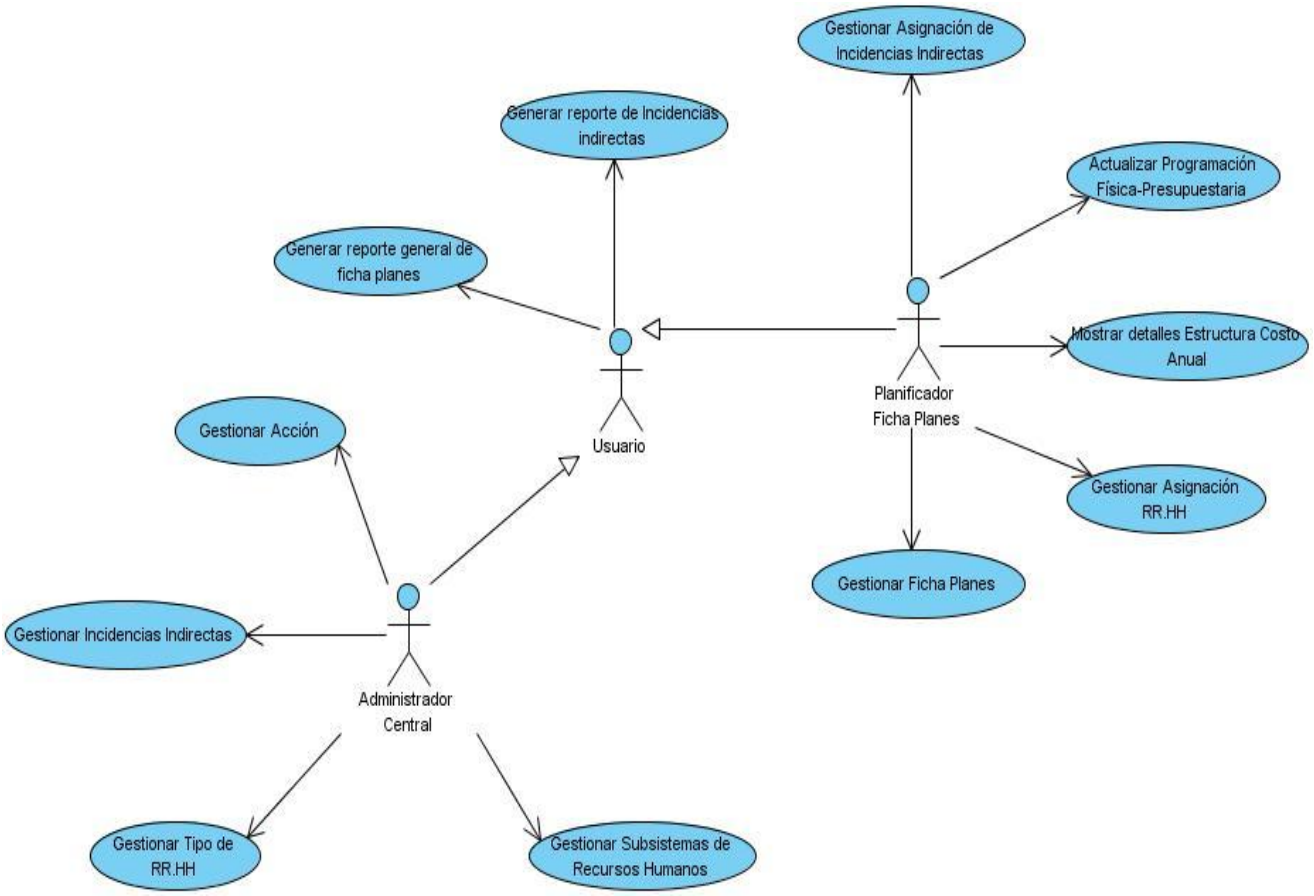


Figura 4 Diagrama de casos de uso del sistema

2.5 DESCRIPCIÓN DE CASOS DE USO DEL SISTEMA

Mediante esta descripción se especifica la secuencia de eventos que los actores utilizan para completar un proceso a través del sistema. Para consultar todas las descripciones de casos de uso, ver **Anexo 3**. A continuación, las descripciones de dos casos de uso del módulo Ficha Planes de Personal:

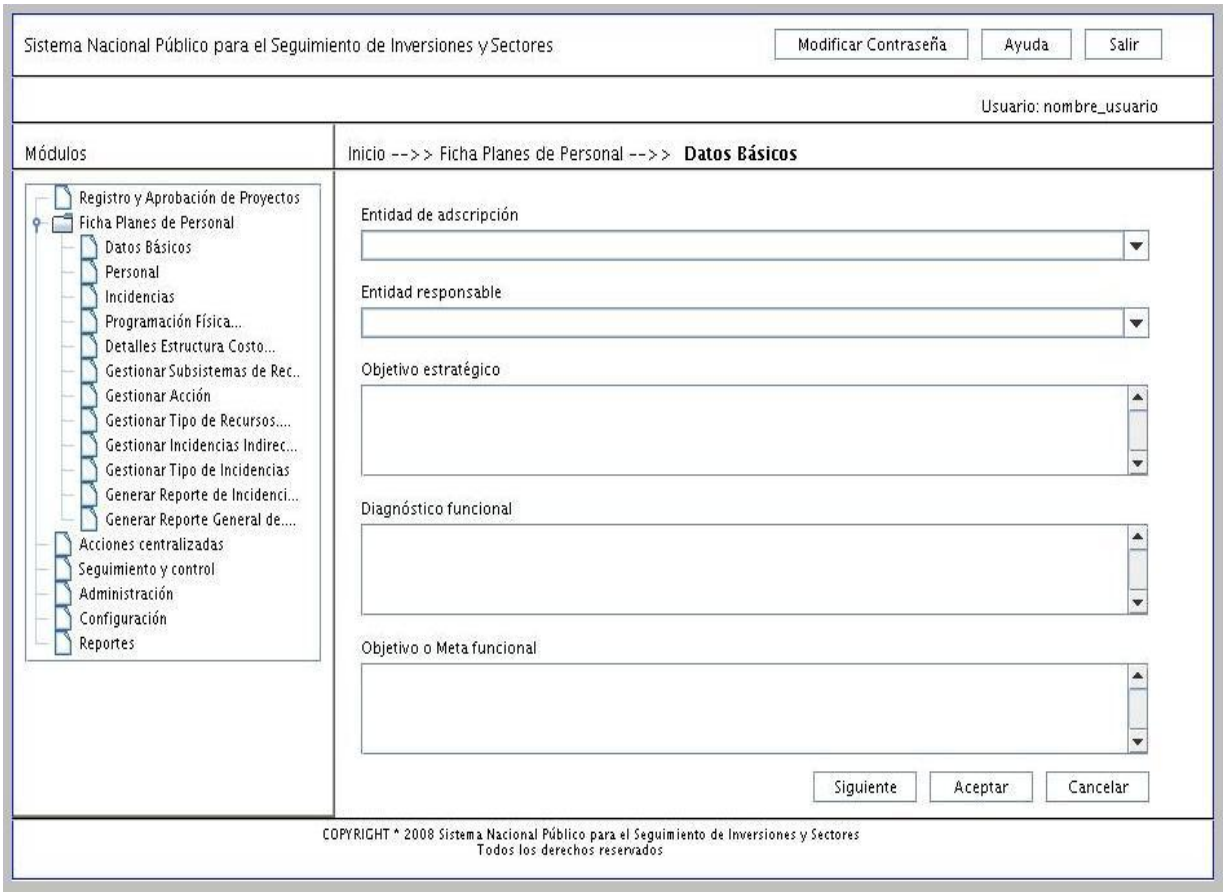
2.5.1 Gestionar Ficha Planes

Caso de Uso:	Gestionar Ficha Planes.
Actores:	Planificador Ficha Planes.
Resumen:	El caso de uso se inicia cuando el Planificador Ficha Planes de una

	entidad necesita gestionar los datos básicos de una Ficha de Planes de Personal. El planificador selecciona la opción correspondiente a los datos básicos de una Ficha Planes de Personal y llena los campos requeridos para la creación en caso que no exista una anteriormente o para la modificación de la existente para esa entidad.
Precondiciones:	• El sistema debe estar instalado y ejecutándose correctamente. • El actor debe estar autenticado con los permisos necesarios.
Referencias	RF_FP.1, RF_FP.2
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 se inicia cuando el actor Planificador Ficha Planes selecciona la opción “Ficha Planes de Personal” y en ella la opción “Datos Básicos”.	2. El sistema muestra la interfaz con los campos necesarios para crear o modificar una Ficha Planes de Personal. En caso que la entidad no tenga ninguna definida hasta el momento los campos aparecen vacíos. En caso que la entidad haya definido una anteriormente aparecen cargados los campos con los datos correspondientes. Los datos que se muestran son los siguientes: • Entidad de adscripción • Entidad responsable • Objetivo estratégico • Diagnóstico funcional • Objetivo o Meta funcional
3. El actor Planificador Ficha Planes completa los campos correspondientes y selecciona la opción “Aceptar”.	4. El sistema valida que los datos introducidos sean correctos y/o que no hay campos obligatorios vacíos.
	5. El sistema guarda/actualiza los datos correspondiente a la Ficha Planes de Personal correspondiente.

Prototipo de Interfaz



Flujos Alternos	
Flujo alterno al paso 3 "Siguiente"	
Acción del Actor	**Respuesta del Sistema**
a. El actor Planificador Ficha Planes selecciona la opción "Siguiente".	b. El sistema invoca al caso de uso Gestionar Asignación RR.HH, terminando así el caso de uso.
Flujo alterno al paso 3 "Cancelar operación"	
Acción del Actor	**Respuesta del Sistema**
a. El actor Planificador Ficha Planes selecciona la opción "Cancelar".	b. El sistema valida que los datos introducidos son correctos y/o que no hay campos obligatorios vacíos.
	c. El sistema cancela la operación, terminando así el caso de uso.
Post-condiciones	El sistema queda con una nueva Ficha Planes de Personal. El sistema queda con una Ficha Planes de Personal actualizada.

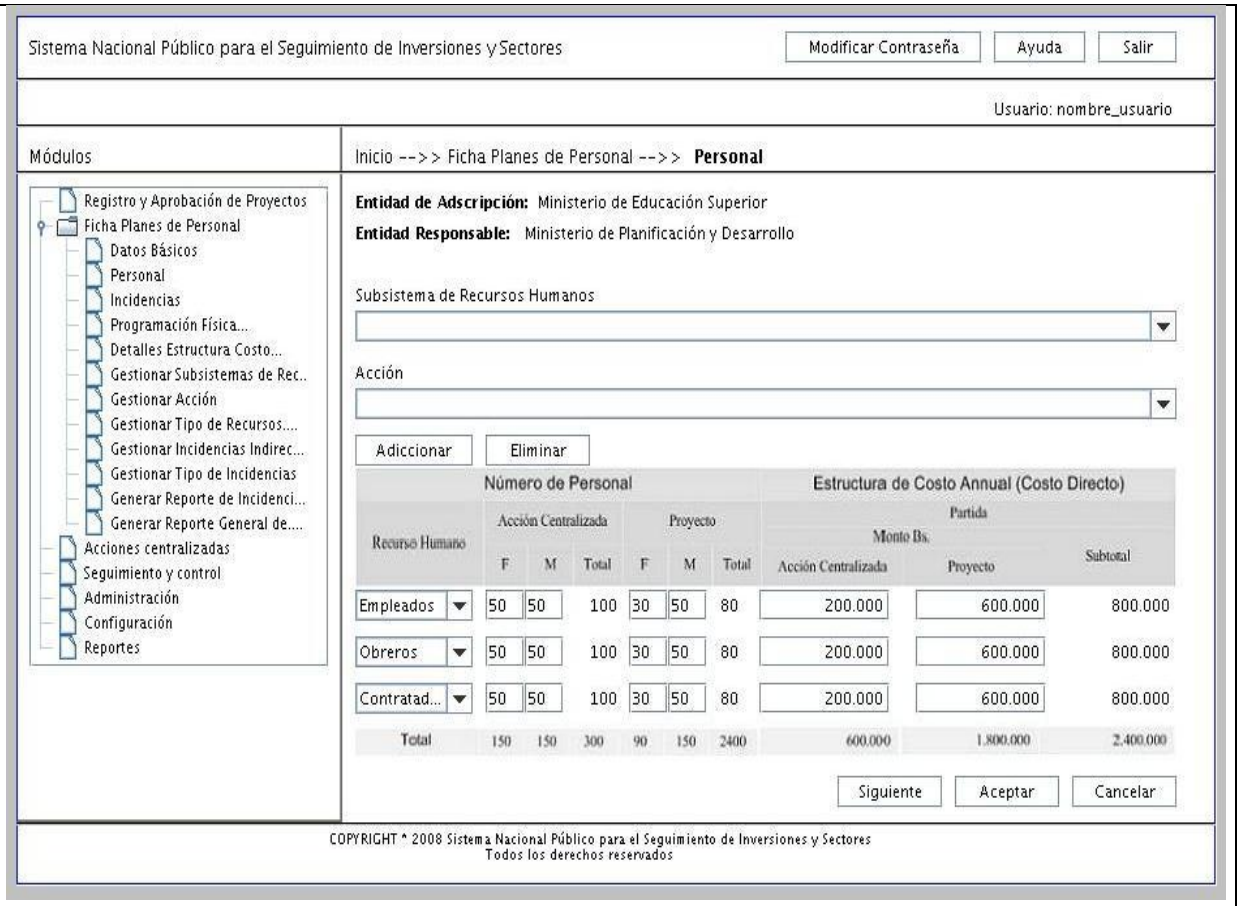
2.5.2 Gestionar Asignación RRHH

Caso de Uso:	Gestionar Asignación RRHH.
Actores:	Planificador Ficha Planes.
Resumen:	El caso de uso se inicia cuando el Planificador Ficha Planes de una entidad necesita gestionar la asignación de una Ficha de Planes de Personal. Este caso de uso puede ser invocado también por el caso de uso Gestionar Ficha Planes. Consiste en que el planificador selecciona la opción correspondiente a Personal de una Ficha Planes de Personal para acceder a las funcionalidades de la gestión de la asignación de Recursos Humanos a una Ficha Planes de Personal.
Precondiciones:	• El sistema debe estar instalado y ejecutándose correctamente. • El actor debe estar autenticado con los permisos necesarios.
Referencias	RF_FP.3, RF_FP.4, RF_FP.5, RF_FP.6
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 se inicia cuando el actor Planificador Ficha Planes selecciona la opción “Ficha Planes de Personal” y en ella la opción “Personal”.	2. El sistema muestra la interfaz donde se lista un conjunto de Recursos Humanos. Los campos que se muestran son los siguientes:
	• Recurso Humano
	• Acción Centralizada
	  o Cantidad personal femenino
	  o Cantidad personal masculino
	  o Total de personal
	  o Monto de costo directo
	• Proyectos
	  o Cantidad personal femenino
	  o Cantidad personal masculino
	  o Total de personal
	  o Monto de costo directo
	• Subtotal de monto directo

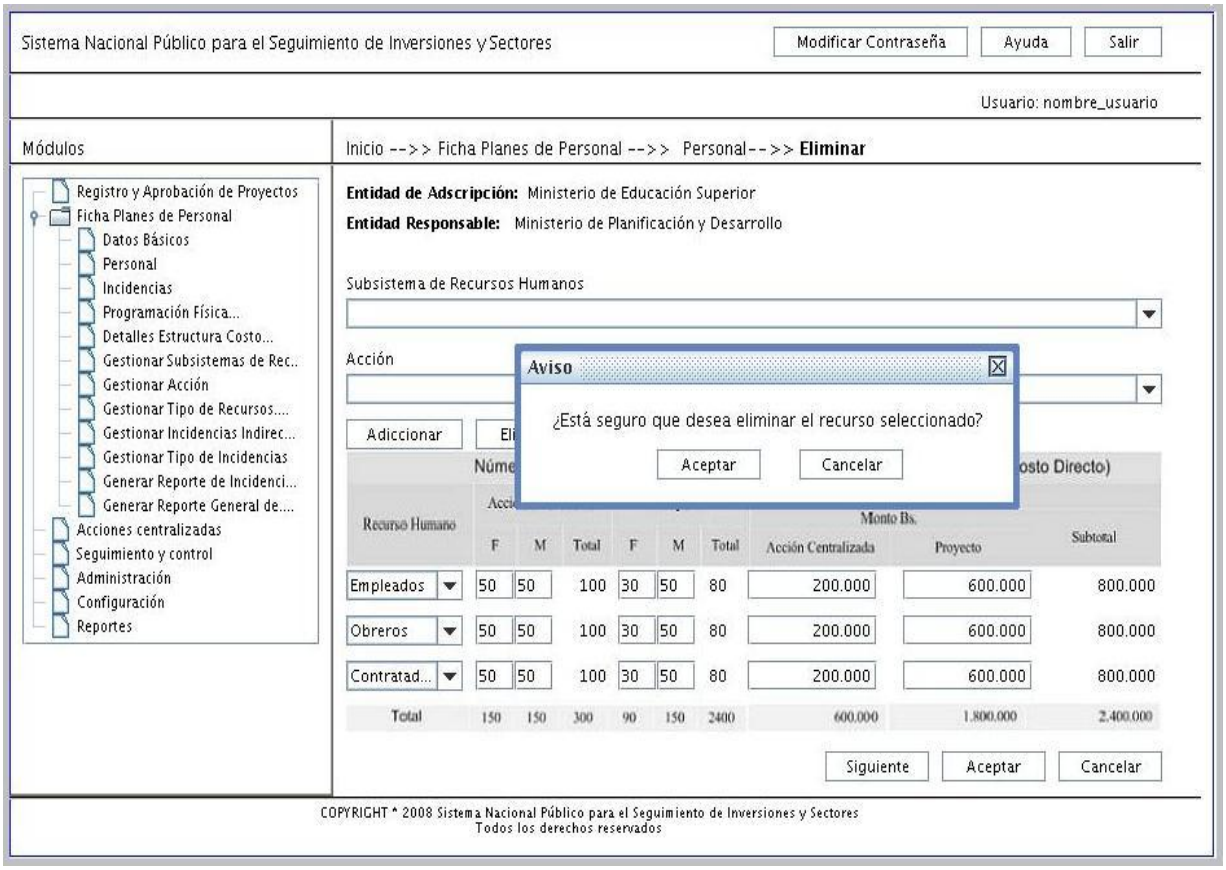
3. El actor Planificador Ficha Planes puede realizar una de las siguientes opciones: • Adicionar ver sección **"Adicionar Recurso Humano"**) • Eliminar (ver sección **"Eliminar Recurso Humano"**)	
Flujo Normal de Eventos	
Sección "Adicionar Recurso Humano"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema adiciona una nueva fila a la lista que se muestra en el paso 2 de la sección anterior con los campos que pueden ser editables. Los campos que se activan como editables son los siguientes: • Recurso Humano • Acción Centralizada o Cantidad personal femenino o Cantidad personal masculino o Monto de costo directo • Proyectos o Cantidad personal femenino o Cantidad personal masculino o Monto de costo directo
2. El actor Planificador Ficha Planes introduce los datos necesarios y presiona la tecla "Enter".	3. El sistema valida que los datos introducidos son correctos y/o que no hay campos obligatorios vacíos.
4. El actor Planificador Ficha Planes selecciona la opción "Aceptar".	5. El sistema registra la nueva asignación de recurso humano. Terminando así el caso de uso.
Prototipo de Interfaz	



Flujos Alternos	
Flujo alterno al paso 4 “Cancelar operación”	
Acción del Actor	**Respuesta del Sistema**
a. El actor Planificador Ficha Planes selecciona la opción “Cancelar”.	b. El sistema cancela la operación, terminando así el caso de uso.
Flujo alterno al paso 4 “Siguiente”	
Acción del Actor	**Respuesta del Sistema**
a. El actor Planificador Ficha Planes selecciona la opción “Siguiente”.	b. El sistema invoca al caso de uso Gestionar Asignación del Incidencias Indirectas.
Sección “Eliminar asignación”	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra una interfaz con un cuadro de diálogo preguntando si está seguro que desea eliminar el elemento seleccionado.
2. El actor Planificador Ficha Planes selecciona la opción “Aceptar” del cuadro	3. El sistema elimina la asignación de recurso humano seleccionada.

de diálogo mostrado.	
4. El actor Planificador Ficha Planes selecciona la opción “Aceptar”.	5. El sistema actualiza la eliminación de recurso humano, terminando así el caso de uso.

Prototipo de Interfaz



Flujos Alternos

Flujo alterno al paso 2,4 “Cancelar operación”

Acción del Actor	Respuesta del Sistema
a. El actor Planificador Ficha Planes selecciona la opción “Cancelar”.	b. El sistema cancela la operación, terminando así el caso de uso.

Flujo alterno al paso 4 “Siguiente”

Acción del Actor	Respuesta del Sistema
a. El actor Planificador Ficha Planes selecciona la opción “Siguiente”.	b. El sistema invoca al caso de uso Gestionar Asignación de Incidencias Indirectas, terminando así el caso de uso.

Post condiciones	El sistema queda con un nuevo recurso humano asignado a una Ficha Planes de Personal. El sistema queda con un recurso humano eliminado de una Ficha Planes de Personal.

2.6 DISEÑO

El diseño es una de las actividades técnicas necesarias para la elaboración del software. Entre las tareas fundamentales del diseño están: producir diseño de datos, diseño arquitectónico, diseño de interfaz y diseño de componentes. El diseño propuesto tiene que cumplir en su totalidad con los requerimientos del sistema, debe ser capaz de facilitar las mejoras del software, y debe especificarse, de forma tal que sea entendible por otros diseñadores y no diseñadores, así como servir de guía para los demás flujos de la ingeniería de software y permitir la comprobación del sistema fácilmente. (28)

En el módulo Ficha Planes de Personal se generan los artefactos siguientes: Subsistema de Diseño, Paquetes de Diseño, Diagrama de Clases del Diseño, Diagramas de Secuencia, Diagrama de Clases Persistentes y Modelo de Datos.

2.7 SUBSISTEMA DE DISEÑO

Los subsistemas de diseño constituyen una forma de estructurar los artefactos que conforman el modelo de diseño en estructuras más independientes. Los elementos del subsistema deben tener un alto grado de cohesión. El subsistema debe tener bajo acoplamiento con respecto a otros subsistemas, lo cual facilita su reutilización, pudiéndose convertir en componentes genéricos, con el fin de separar los aspectos de diseño y lograr una mayor independencia en cuanto a desarrollo y a reutilización de funcionalidades. A continuación se presentan los subsistemas de diseño relacionados con el módulo Ficha Planes de Personal del sistema SINAPSIS.

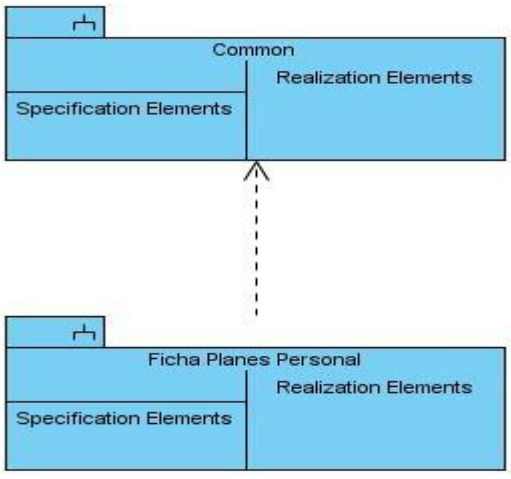


Figura 5 Subsistemas de diseño del módulo Ficha Planes de Personal

2.8 PAQUETES DE DISEÑO

Los paquetes de diseño permiten la agrupación de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes relacionados, utilizados como contenedores de elementos. A continuación se presentan los paquetes identificados en el módulo Ficha Planes de Personal que facilitan la organización de los elementos del diseño.

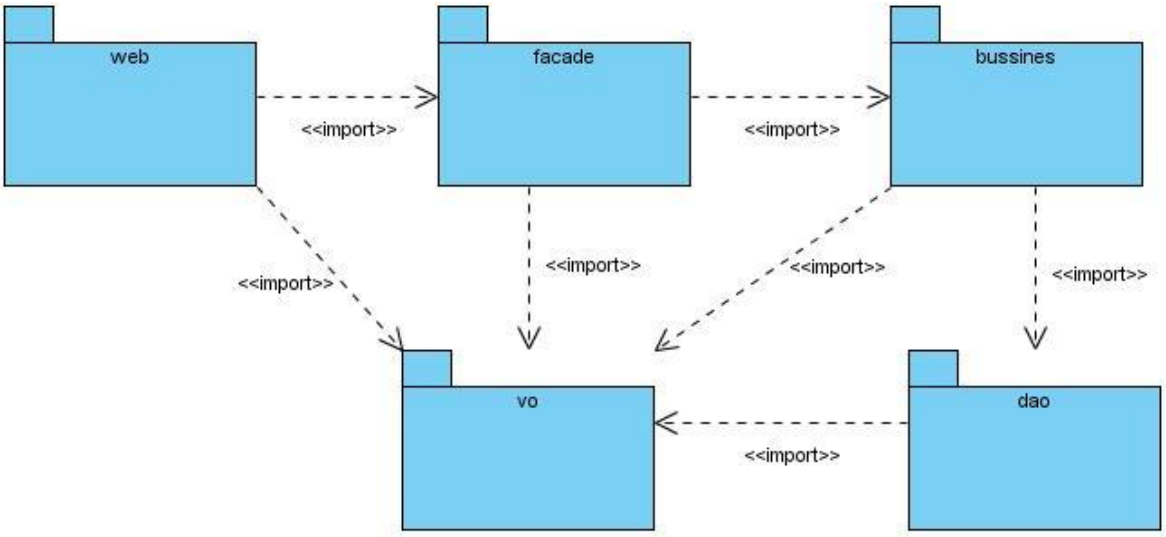


Figura 6 Paquetes de diseño del módulo Ficha Planes de Personal

2.9 DIAGRAMA DE CLASES DEL DISEÑO

Los diagramas de clases del diseño describen la realización de los casos de uso y a la vez constituyen una abstracción del modelo de implementación y del código fuente. Son una entrada esencial a las actividades de implementación. Como el sistema SINAPSIS es una aplicación web, en los diagramas de clases de diseño elaborados se hizo uso de los estereotipos web, para ilustrar la colaboración real y la representación de los elementos que interactúan en la ejecución de las funcionalidades que debe brindar el sistema.

2.9.1 Patrones utilizados para el diseño de las clases

En el diseño de las clases se usaron los patrones de diseño estudiados en el Capítulo 1. Teniendo en cuenta que la implementación del sistema se realizará en Java con el uso del *Framework Spring*, se diseñaron las clases respetando una estructura por capas que permite la implementación del patrón *Factory Method* utilizado en *Spring*. Para la comunicación entre capas se utilizó el patrón *Facade,* y el DAO para crear una interfaz común que garantice el acceso a los datos. (30) Además se respetaron los estándares de diseño, código y nombrado de las clases definidos por los Arquitectos de SINAPSIS. A

continuación se muestran los diagramas de clases de diseño de los casos de uso principales del módulo Ficha Planes de Personal.

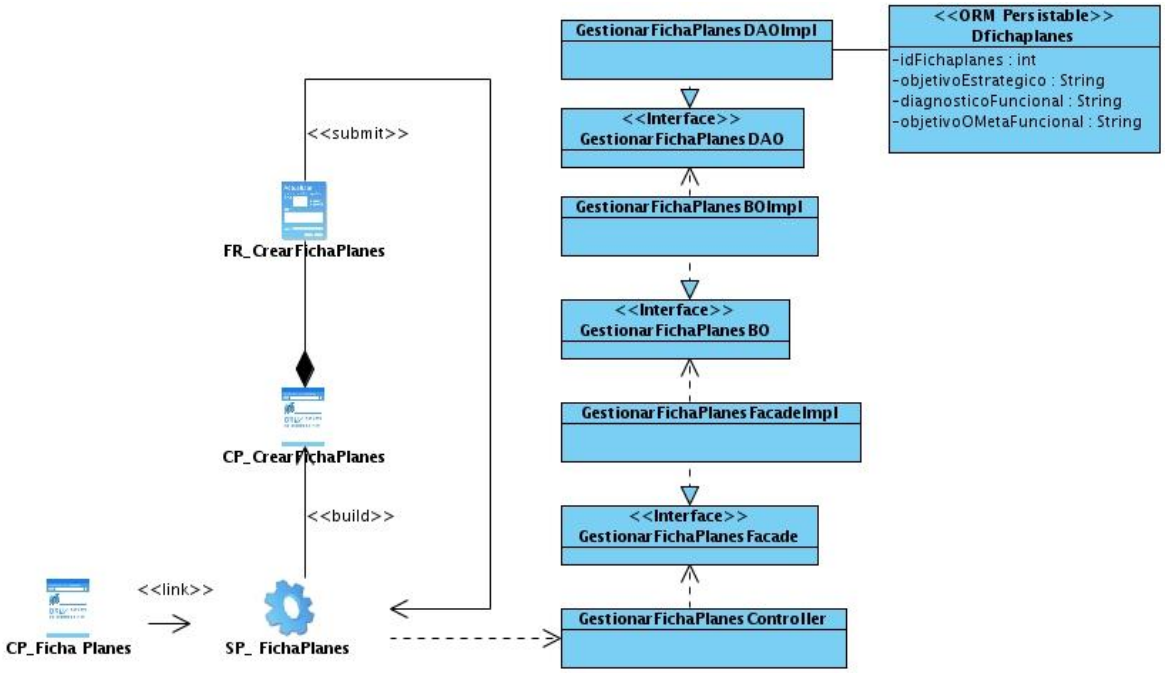


Figura 7 Diagrama de clases de diseño, CU Gestionar Ficha Planes.

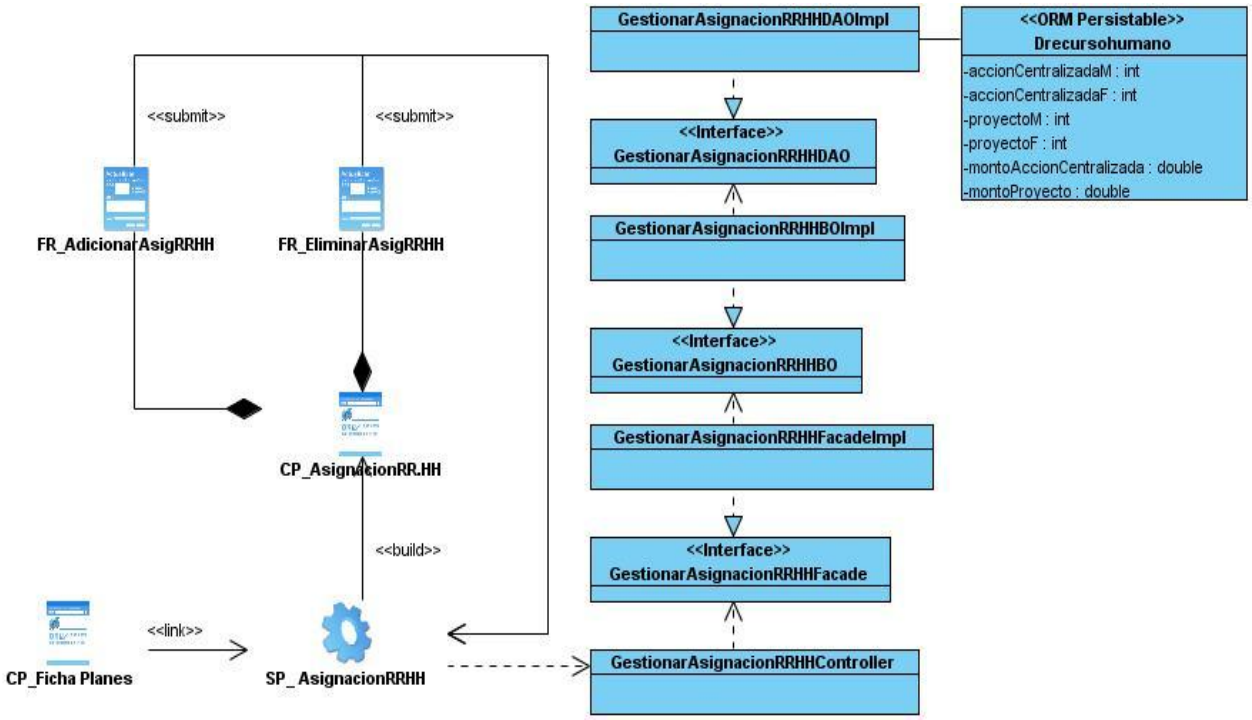


Figura 8 Diagrama de clases de diseño, CU Gestionar Asignación RR.HH.

Para consultar todos los diagramas de clases del diseño, ver **Anexo 4**.

2.10 DIAGRAMAS DE SECUENCIA

Los diagramas de secuencia muestran una interacción ordenada según la secuencia temporal de eventos y, en particular, los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo. A los casos de uso que tienen varias secciones se les realizó un diagrama de secuencia por cada flujo para un mejor entendimiento. A continuación se muestra el diagrama de secuencia para un escenario del caso de uso “Gestionar Asignación RR.HH” del módulo Ficha Planes de Personal.

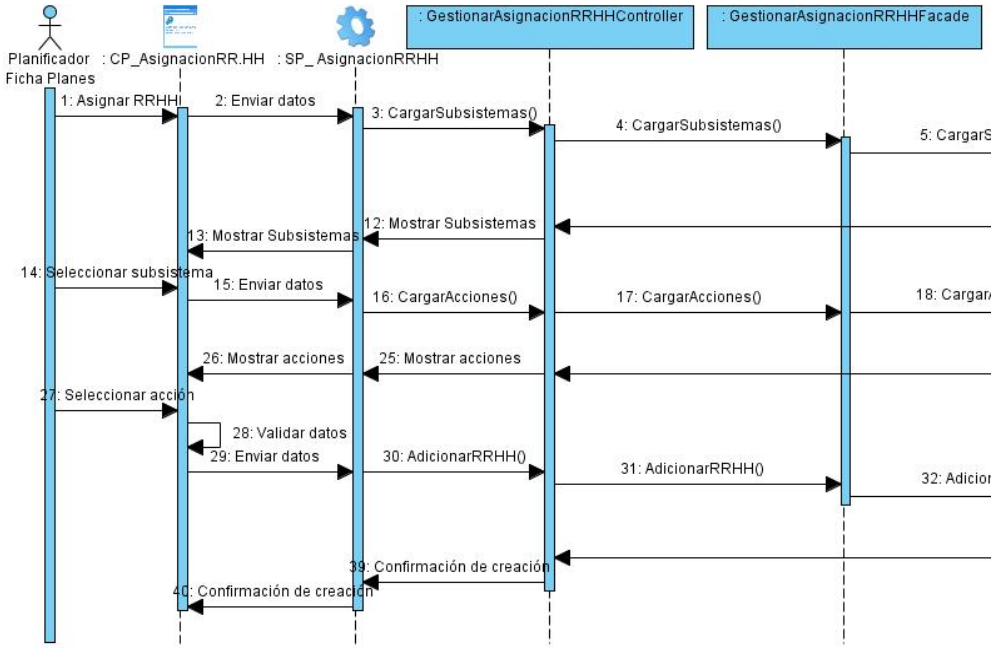


Figura 9 Diagrama de Secuencia: CU Gestionar Asignación RR.HH, Sección Adicionar.

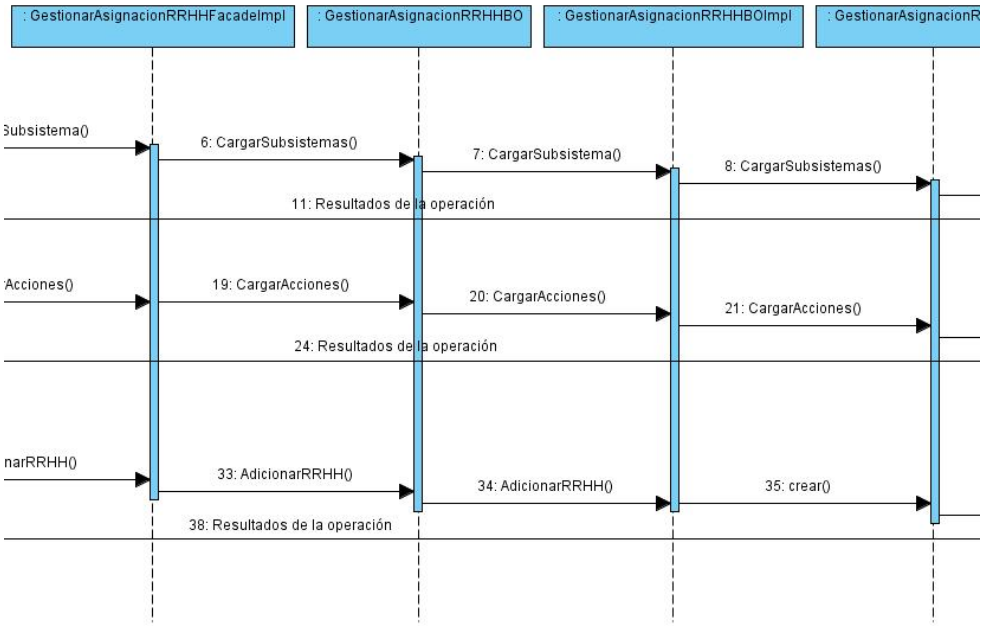


Figura 10 Diagrama de Secuencia: CU Gestionar Asignación RR.HH, Sección Adicionar.

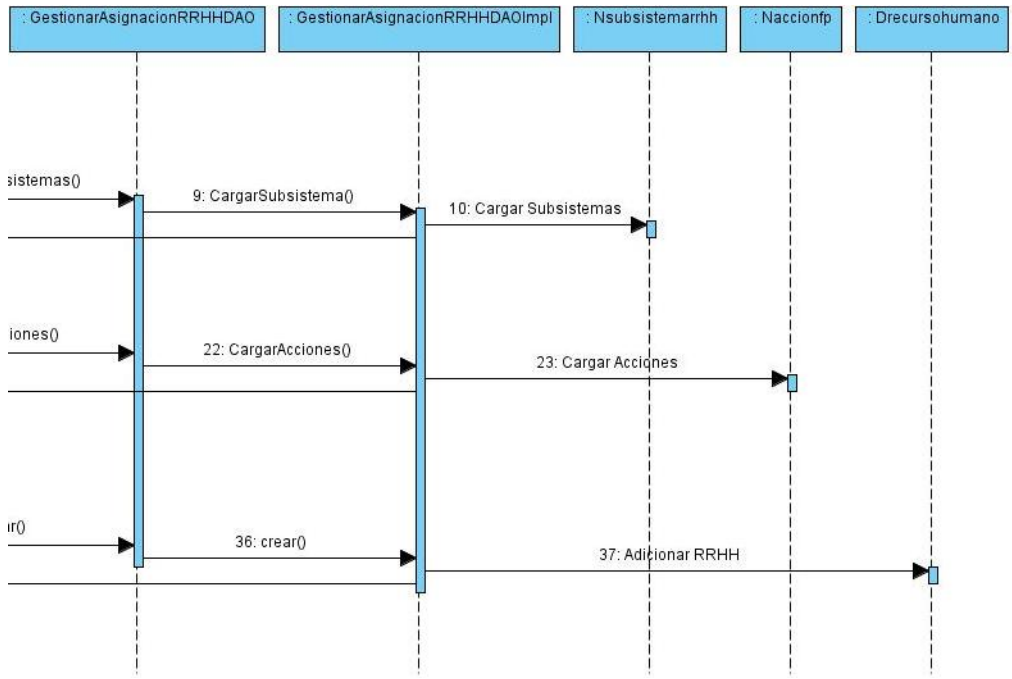


Figura 11 Diagrama de Secuencia: CU Gestionar Asignación RR.HH, Sección Adicionar.

Para consultar todos los diagramas de secuencias, ver **Anexo 4.**

2.11 DIAGRAMA DE CLASES PERSISTENTES

Para facilitar la persistencia de los datos y poder generar el modelo de datos que utilizará el sistema de base de datos y las tablas de la base de datos, se generó el diagrama de clases persistentes, el cual representa una relación entre los objetos persistentes y sus respectivas relaciones, dependencias y cardinalidades.

Para consultar el diagrama de clases persistentes, ver **Anexo 4**.

2.12 MODELO DE DATOS

Es un modelo abstracto que describe cómo deben ser representados y usados los datos. Sirve para describir la estructura de la base de datos, los datos, sus relaciones y las restricciones que deben cumplirse entre ellos.

Para consultar el modelo de datos, ver **Anexo 4.**

2.13 CONCLUSIONES

En este capítulo se llegó a las siguientes conclusiones:

- La utilización de técnicas para la identificación de requerimientos, permitió obtener resultados satisfactorios en la identificación de los requerimientos del sistema.
- Se logró un refinamiento de los requerimientos capturados, se identificaron los actores encargados de interactuar con el sistema, y se generó el diagrama de casos de uso del sistema y su especificación.
- Los artefactos creados en Visual Paradigm y UML como lenguaje de modelado, propició un mayor entendimiento entre los involucrados en el módulo Ficha Planes de Personal del sistema SINAPSIS.
- La aplicación de patrones, tanto de casos de uso como de diseño, posibilitó obtener artefactos aceptables.
- La construcción del modelo de casos de uso del sistema y de diseño permitió obtener los elementos que deberán ser implementados.

CAPÍTULO 3

3.1 INTRODUCCIÓN

En el presente capítulo se analizan los resultados obtenidos durante el desarrollo de la solución propuesta. Para la validación de los artefactos del modelo de casos de uso del sistema, se muestran las listas de chequeo que se le aplicaron, además del acta de liberación. Se aplican métricas para medir la calidad de la funcionalidad del diagrama de casos de uso del sistema. Para la validación de los artefactos del modelo del diseño se aplican métricas a nivel de componentes y orientadas a clases.

3.2 VALIDACIÓN DE LOS REQUERIMIENTOS

La identificación de los requerimientos y la especificación de los casos de uso, constituyen los principales artefactos obtenidos durante el desarrollo de la ingeniería de requerimientos del módulo Ficha Planes de Personal del sistema SINAPSIS. Con el objetivo de garantizar su factibilidad, fueron sometidos a varias iteraciones de revisiones por parte del equipo de calidad a nivel de proyecto y facultad. Se aplicaron listas de chequeo para validar los requerimientos (**Anexo 5**). Todas las no conformidades encontradas durante las iteraciones de la revisión fueron solucionadas por el equipo de análisis del proyecto. Para consultar el acta de liberación de la documentación revisada, ver **Anexo 6**.

3.3 MÉTRICAS PARA LA CALIDAD DE LA FUNCIONALIDAD DEL DIAGRAMA DE CASOS DE USO DE SISTEMA

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

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

Estos atributos presentan un significado determinado de acuerdo con el tipo de artefacto y el nivel de abstracción que éste describe. Cada atributo se evalúa en términos de un conjunto de factores, los cuales tendrán asociados una métrica.

A continuación se muestra la tabla de las métricas correspondientes a cada uno de los atributos especificados anteriormente:

No.	Atributo	Métricas
1.	Completitud	Número de casos de uso que no tienen descripción resumida.
2.		Número de casos de uso que tienen requerimientos omitidos.
3.		Número de casos de uso que no poseen una descripción extendida.
4.		Número de casos de uso que tienen acciones del flujo de eventos no redactados en función del responsable.
5.		Número de casos de uso que no describen condiciones de excepción relevantes.
6.		Número de casos de uso que no han sido clasificados.
7.	Consistencia	Número de casos de uso que tienen un nombre incorrecto.
8.		Número de casos de uso que no representan una interacción observable por un actor.
9.		Número de casos de uso que tienen acciones del flujo de eventos asignados a un responsable que no le corresponde.
10.		Número de casos de uso no aceptados.
11.		Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos.
12.		Número de casos de uso que no tienen un usuario responsable.
13.	Correctitud	Número de casos de uso en que los requerimientos representados no son comprensibles por el usuario.
14.		Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema.

15.		Número de casos de uso que deben ser modificados para mejorar el proceso actual.
16.	Complejidad	Número de elementos del diagrama que requieren reubicación.

Resultados por métricas

Métrica	No. de casos de uso analizados	Resultado	Porcentaje de error
1	11	Todos poseen una descripción resumida. Después de varias revisiones en busca de deficiencias, se llegó al presente estado donde no existe ninguna.	0%
2	11	Ninguno presenta requerimientos omitidos, al menos están relacionados con un requerimiento.	0%
3	11	Se detectó un caso de uso que no presentaba su descripción extendida.	9.09%
4	11	Todos están redactados en función del responsable que le corresponde a cada uno.	0%
5	11	No se detectó ningún caso de uso que no describiera condiciones de excepción relevantes.	0%
6	11	Todos han sido clasificados según los diferentes tipos de prioridad.	0%
7	11	Ninguno presentó problemas en cuanto al nombre dado, ya que todos los nombres proporcionados a cada uno de los casos de uso representaban una expresión verbal en infinitivo describiendo alguna funcionalidad relevante y significativa para el usuario.	0%

8	11	Todos representan correctamente la interacción con un actor.	0%
9	11	Ninguno presentó acciones del flujo de eventos que estuvieran asignados a un responsable que no le corresponde.	0%
10	11	Todos fueron aceptados.	0%
11	11	Todos poseen sus descripciones del flujo básico y de flujos alternos separadas.	0%
12	11	Todos poseen un usuario responsable.	0%
13	11	Todos los casos de uso en que los requerimientos fueron representados son comprensibles por el usuario.	0%
14	11	Se detectó un caso de uso que debe ser modificado para adecuarlo a la funcionalidad del sistema.	9.09%
15	11	Ninguno tuvo que ser modificados para mejorar el proceso actual.	0%
16		Ninguno de los elementos que conforman el diagrama de casos de uso requiere reubicación.	0%

Luego de haber aplicado las métricas al diagrama de casos de uso, se han graficado los resultados obtenidos, los cuales se representan a continuación:

Figura 12 Resultados de la evaluación de la calidad en los diagramas de casos de usos.

3.4 MÉTRICAS PARA LA VALIDACIÓN DEL DISEÑO

3.4.1 Métricas de diseño arquitectónico

Complejidad Estructural

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

$S(i)$: Complejidad estructural

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

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

Complejidad de Datos

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

$D(i)$: Complejidad en la interfaz interna del módulo Ficha Planes de Personal.

$V(i)$: Número de variables de entrada y salida, que entran y salen del módulo Ficha Planes de Personal.

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

$$D(i) = \frac{10}{2 + 1} = 3.33$$

Complejidad del sistema

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

$C(i)$: Complejidad del sistema.

$C(i) = 4 + 3.33 = 7.33$

Generalmente se proponen tres tipos de sistema según su complejidad:

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

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

3.4.2 Métricas de diseño a nivel de componente

Métricas de cohesión

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

CFD: Cohesión funcional débil.

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.

Se analizaron elementos pertenecientes a los principales conceptos que plantea la métrica, dígase Porción de datos, Símbolos léxicos (*tokens*) de datos, Señales de unión, Señales de súper-unión y Cohesión a partir de los datos recogidos en la siguiente tabla.

Clases	Usabilidad
GenerarReporteFichaFacade	1
GestionarAsignacionRR.HHController	1
BaseDAO	11
GestionarFichaPlanesBO	2
GestionarAsignacionRR.HHBO	3
GestionarAsignacionIncIndBO	3
ActProgramacionFPBO	2
MostrarDetallesECABO	2
GestionarSubsistemasRRHHBO	1
GestionarAccionBO	2
GestionarIncIndBO	2
GenerarReporteIncIndBO	4

Según los datos de las clases analizadas se tiene que:
número de elementos = 12
número de súper adhesivos(i) = 0
número de adhesivos(i) = 9

$$CFD = \frac{\text{número de adhesivos (i)}}{\text{número de elementos (i)}}$$

$$CFD = \frac{9}{12} = 0.75$$

La métrica de Bieman y Ott plantea que mientras más cerca están los valores de CFD de 1, mayor será la cohesión del módulo. Los resultados demuestran que el valor de CFD es alto. La relación del número de clases adhesivas con el número total de elementos de la

muestra determina que el subsistema de diseño ¨Ficha Planes de Personal¨ del sistema SINAPSIS posee una cohesión funcional alta para un 75% de fortaleza.

3.4.3 Métricas orientadas a clases

Tamaño de clase (TC)

Para medir el tamaño de las clases (TC), se tienen en cuenta los aspectos siguientes:

- Total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.
- Cantidad de atributos (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase.
- Promedio general de los dos anteriores para el sistema completo.

Para evaluar las métricas son necesarios los valores de los umbrales. Las clases se clasifican en tres grupos, según su tamaño, los que se presentan en la siguiente tabla junto con los umbrales seleccionados para su clasificación.

Clasificación	**Valores de los Umbrales**
Pequeño	<=20
Medio	>20 y<=30
Grande	>30

Esta métrica se realizó con las clases de mayor peso y prioridad dentro del Módulo Ficha Planes de Personal del sistema SINAPSIS.

Clases	**No. Atributos**	**No. Operaciones**
GestionarFichaPlanesDAOImpl	2	10
GestionarAsignaciónRR.HHDAOImpl	2	15
GestionarSubsistemasRRHHDAOImpl	2	12
GestionarFichaPlanesBOImpl	4	2
GestionarAsignaciónRR.HHBOImpl	4	6
GestionarSubsistemasRRHHBOImpl	4	5

GestionarFichaPlanesFacadeImpl	1	2
GestionarAsignaciónRR.HHFacadeImpl	1	6
GestionarSubsistemasRRHHFacadeImpl	1	5
GestionarFichaPlanesController	3	2
GestionarAsignaciónRR.HHController	3	6
GestionarSubsistemasRRHHController	3	5

En las 12 clases presentadas se obtuvo un promedio de 2.5 atributos y 6.33 operaciones o métodos por clase. Los resultados arrojados por la métrica son:

Umbral	Tamaño	Cantidad de Clases
<=20	Pequeño	12
>20 y <=30	Mediano	0
>30	Grande	0

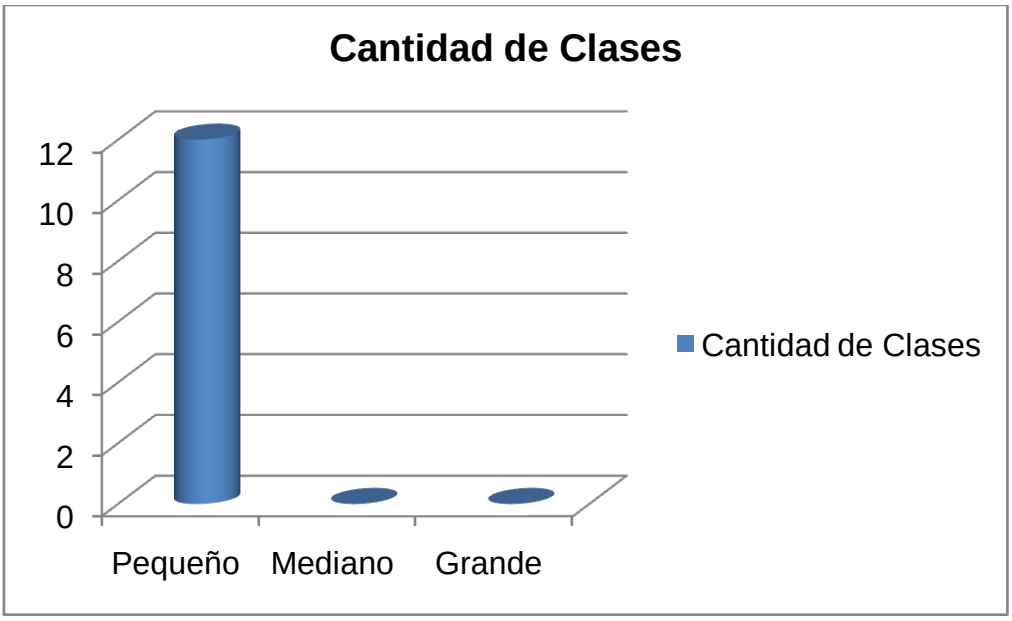


Figura 13 Número de clases por categorías.

Atendiendo a la cantidad de operaciones, el 100 % de las clases diseñadas están consideradas como pequeñas.

Árbol de Profundidad de Herencia (APH)

Esta métrica se define como la longitud máxima desde el nodo hasta la raíz del árbol de herencia (Capítulo 1).

A continuación se establecen los valores de profundidad de herencia para las clases de mayor peso y prioridad dentro del Módulo Ficha Planes de Personal del sistema SINAPSIS.

Clases	PH
GestionarFichaPlanesDAOImpl	1
GestionarAsignaciónRR.HHDAOImpl	1
GestionarSubsistemasRRHHDAOImpl	1
GestionarFichaPlanesBOImpl	0
GestionarAsignaciónRR.HHBOImpl	0
GestionarSubsistemasRRHHBOImpl	0
GestionarFichaPlanesFacadeImpl	0
GestionarAsignaciónRR.HHFacadeImpl	0
GestionarSubsistemasRRHHFacadeImpl	0
GestionarFichaPlanesController	0
GestionarAsignaciónRR.HHController	0
GestionarSubsistemasRRHHController	0

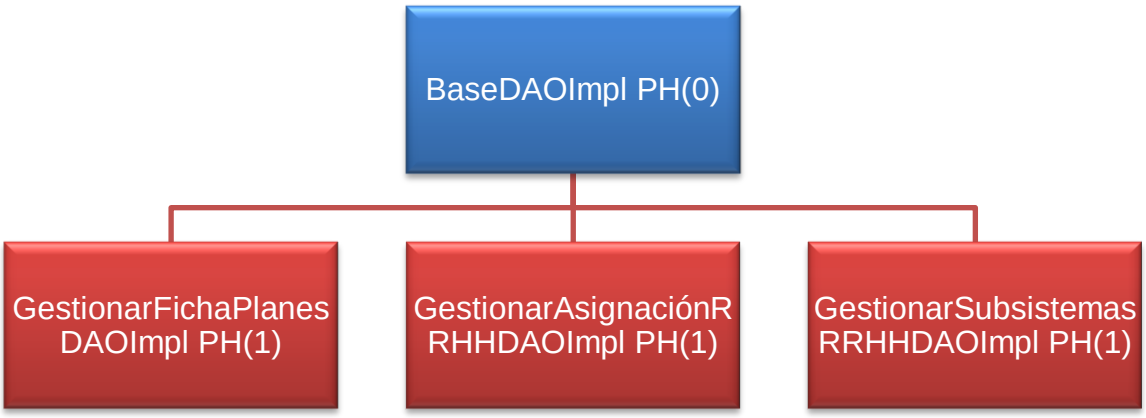


Figura 14 Árbol de Profundidad de la Herencia del módulo.

De acuerdo con los datos obtenidos, luego de aplicar al módulo la métrica APH, se obtuvo un valor de 1 como el nivel más alto de herencia, 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.

3.5 CONCLUSIONES

Se puede concluir que:

- El acta de liberación por parte del equipo de calidad de la facultad demostró que tanto los requerimientos como la especificación de los casos de uso del sistema quedaron bien definidos, especificados y sin ambigüedad, abarcando las necesidades del cliente.
- La aplicación de las métricas para la calidad de la funcionalidad del diagrama de casos de uso del sistema demostró que se construyó un diagrama con calidad.
- Las métricas aplicadas a los elementos del diseño del módulo validan la calidad del diseño en cuanto a complejidad y cohesión.
- El subsistema de diseño ¨Ficha Planes de Personal¨ del sistema SINAPSIS posee una cohesión funcional alta para un 75 % de fortaleza.
- El 100 % de las clases diseñadas están consideradas como pequeñas, lo que facilitará el proceso de construcción del módulo Ficha Planes de Personal.

CONCLUSIONES GENERALES

Una vez concluido el desarrollo del presente trabajo se puede llegar a las siguientes conclusiones:

- El estudio del arte de las diferentes metodologías y herramientas permitieron definir cuáles eran las adecuadas para implementar en el desarrollo del sistema.
- El análisis de los procesos de negocio, junto a la interacción directa con los clientes del sistema, permitió que se obtuvieran resultados satisfactorios en la identificación de los requerimientos que debe cumplir el sistema.
- La elaboración de los artefactos del modelo de casos de uso del sistema posibilitó un entendimiento común entre desarrolladores y clientes, en cuanto a las funcionalidades que el sistema debe brindar.
- La construcción de los artefactos del modelo de diseño facilitó obtener los elementos que deberán ser implementados, para que el sistema cumpla requerimientos especificados.
- Los artefactos obtenidos presentan resultados positivos, tomando como referencia la aplicación de métodos y métricas para validar la solución del mismo.

RECOMENDACIONES

Con vistas al desarrollo del sistema SINAPSIS y la implementación del Módulo Ficha Planes de Personal, se recomienda:

- Seguir utilizando RUP como metodología de desarrollo de software para generar los artefactos correspondientes a los siguientes flujos de trabajo definidos por este proceso.
- Realizar un seguimiento de los requerimientos de software durante las posteriores fases de desarrollo, aplicando la Administración de requerimientos que propone la Ingeniería de Requerimientos.
- Continuar con la elaboración de los restantes artefactos que genera el flujo de requerimiento y diseño, que facilitan continuar con el desarrollo del módulo.
- Realizar la implementación del módulo Ficha Planes de Personal del sistema SINAPSIS.

BIBLIOGRAFÍA

1. **Centro Experimental de Ingeniería de Software.** CEIS. (s.f.). [Online] Diciembre 13, 2007. http://www.ceis.cl/Gestacion/Gestacion.

2. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000. pp. 5-8.

3. —. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000. p. XVI.

4. **Rational Software Corporation.** Rational Unified Process. [Online] 2003. http://www.rational.com.

5. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000. pp. 11-16.

6. **Cortizo Pérez, José Carlos, Expósito Gil, Diego and Ruiz Leyva, Miguel.** *eXtreme Programming.* España : s.n., 2004.

7. **Microsoft Corporation.** MSF Process Model v. 3.1. [Online] Junio 2002. http://www.microsoft.com/msf/.

8. **Universidad Simón Bolivar.** LISI Laboratorio de informacion en sistemas informaticos . [Online] 1999. http://www.lisi.usb.ve/publicaciones/07%20integracion%20de%20sistemas/integracion_23.pdf.

9. Rational Rose. [Online] Diciembre 6, 2007. http://www.rational.com.

10. **SPARX System.** SPARX System Pty L tb. [Online] 2000-2007. http://sparxsystems.com.ar/products/ea.html.

11. Visual Paradigm. [Online] Diciembre 6, 2007. http://www.visual-paradigm.com/.

12. Axure-rp. [Online] http://axure-rp.softonic.com/.

13. **Microsoft Corporation.** *Información general del producto Microsoft Office Visio 2007.* 2006.

14. **Microsoft Corporation.** Microsoft Office Visio Homepage. [Online] Abril 5, 2009. http://office.microsoft.com/es-es/visio/FX100487863082.aspx.

15. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El proceso unificado de desarrollo de software.* s.l. : Addison Wesley, 2000.

16. **Mesa, Francisco Aragón.** *Introducción a la programación orientada a objetos.*

17. **W3C World Wide Web, Consortium.** W3C World Wide Web. [Online] 1994.

18. **Young, Ralph R.** *The Requirements Engineering Handbook.* Londres : s.n., 2004.

19. **IEEE Computer Dictionary.** *Compilation of IEEE Standard Computer Glossaries.* 1990.

20. **Kontoya, G and Sommerville, I.** *Requirements Engineering: Processes and Techniques.* 2000.

21. **Pressman, Rogen S.** *Ingeniería de Software. "Un enfoque práctico".* s.l. : Addison Wesley, 2005.

22. **IEEE, SWEBOK.** *Guide to the Software Engineering Body of Knowledge.* 2004.

23. **Escalona, José María and Koch, Nora.** *Ingeniería de Requerimientos en Aplicaciones para la Web - Un estudio comparativo.* 2002.

24. **Antonelli, Leandro y Oliveros, Alejandro.** *Revisión de las fuentes usadas para elicitar requerimientos. .* 2003.

25. **Sumano, María de los Ángeles.** *Análisis de Requerimientos de Software. Estado del Arte.* 1999.

26. **Wiegers, Karl E.** *More About Software Requirements: Thorny Issues and Practical Advice.* 2006. 0735622671.

27. **IEEE Std.** *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications –Description.* 1998.

28. **Pressman, Rogen S.** *Ingeniería de Software "Un enfoque práctico".* Quinta. Madrid : Graw Hill, 2002.

29. **Gunnar Övergaard, Karin Palmkvist.** *Use Cases Patterns and Blueprints.* Addison Wesley : s.n., 2004.

30. **E, Gamma, et al.** *Design Patterns. Elements of Reusable Object-Oriented.* s.l. : Adisson Wesley, 2003.

31. **Adriano, Natalia.** *Comparación del Proceso de Elicitación de Requerimientos en el desarrollo de Software a Medida y Empaquetado. Propuesta de métricas para la elicitación.* 2006.

ANEXOS

ANEXO 1. DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS FUNCIONALES

Este documento contiene las condiciones o capacidades que el sistema debe cumplir con sus respectivas especificaciones.
(Consultar documento con el nombre ***"D. Especificación de Requerimientos.pdf"*** en la carpeta Artefactos).

ANEXO 2. DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS NO FUNCIONALES

Este documento contiene las propiedades o cualidades que el sistema deberá tener.
(Consultar documento con el nombre ***"D. Especificación de Requerimientos No Funcionales.pdf"*** en la carpeta Artefactos).

ANEXO 3. MODELO DE SISTEMA FICHA PLANES DE PERSONAL

Este documento contiene todos los actores del sistema y su relación con los Casos de uso del sistema, además de la descripción de cada caso de uso.
(Consultar documento con el nombre ***"F. Modelo de Sistema Ficha Planes.pdf"*** en la carpeta Artefactos).

ANEXO 4. MODELO DE DISEÑO FICHA PLANES DE PERSONAL

Este documento contiene la realización física de los casos de usos centrándose en cómo los requerimientos funcionales y no funcionales tienen impacto en el sistema a considerar
(Consultar documento con el nombre ***"Modelo de diseño.pdf"*** en la carpeta Artefactos).

ANEXO 5. LISTAS DE CHEQUEO PARA VALIDAR LOS REQUERIMIENTOS

Este documento contiene las listas de chequeo que fueron aplicadas a los requerimientos del módulo Ficha Planes de Personal.
(Consultar documento con el nombre ***"Listas de Chequeo.pdf"*** en la carpeta Artefactos).

ANEXO 6. ACTA DE LIBERACIÓN DEL GRUPO DE CALIDAD

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 Ficha Planes de Personal del proyecto Sinapsis del Centro CEGEL de la Facultad 15 y haberse detectado un promedio de 21 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

GLOSARIO DE TÉRMINOS

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

CASE *(Computer Aided Software Engineering):* Ingeniería de Software Asistida por Ordenador.

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

CRUD (*Create, Read, Update y Delete*): patrón de casos de uso. Plantea que las funcionalidades de crear, obtener, actualizar y eliminar se pueden agrupar en un mismo CU.

Ingeniería de Requerimientos: 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”.

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

Metodologías: es una palabra compuesta por tres vocablos griegos: metà (**más allá**), odòs (**camino**) y logos (**estudio**). Son los métodos de investigación que permiten lograr ciertos objetivos en una ciencia. En otras palabras, la metodología es una etapa específica que procede de una posición teórica y epistemológica, para la selección de técnicas concretas de investigación.

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

Módulo Ficha Planes de Personal: es la parte de sistema que se encarga de la automatización del proceso de control de recursos humanos y materiales para la ejecución de los proyectos.

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

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.

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

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

RUP (*Rational Unified Process*): se trata de una metodología para describir el proceso de desarrollo de software.

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 (*Unified Modeling Language*): es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.