Universidad de las Ciencias Informáticas

Facultad 3

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

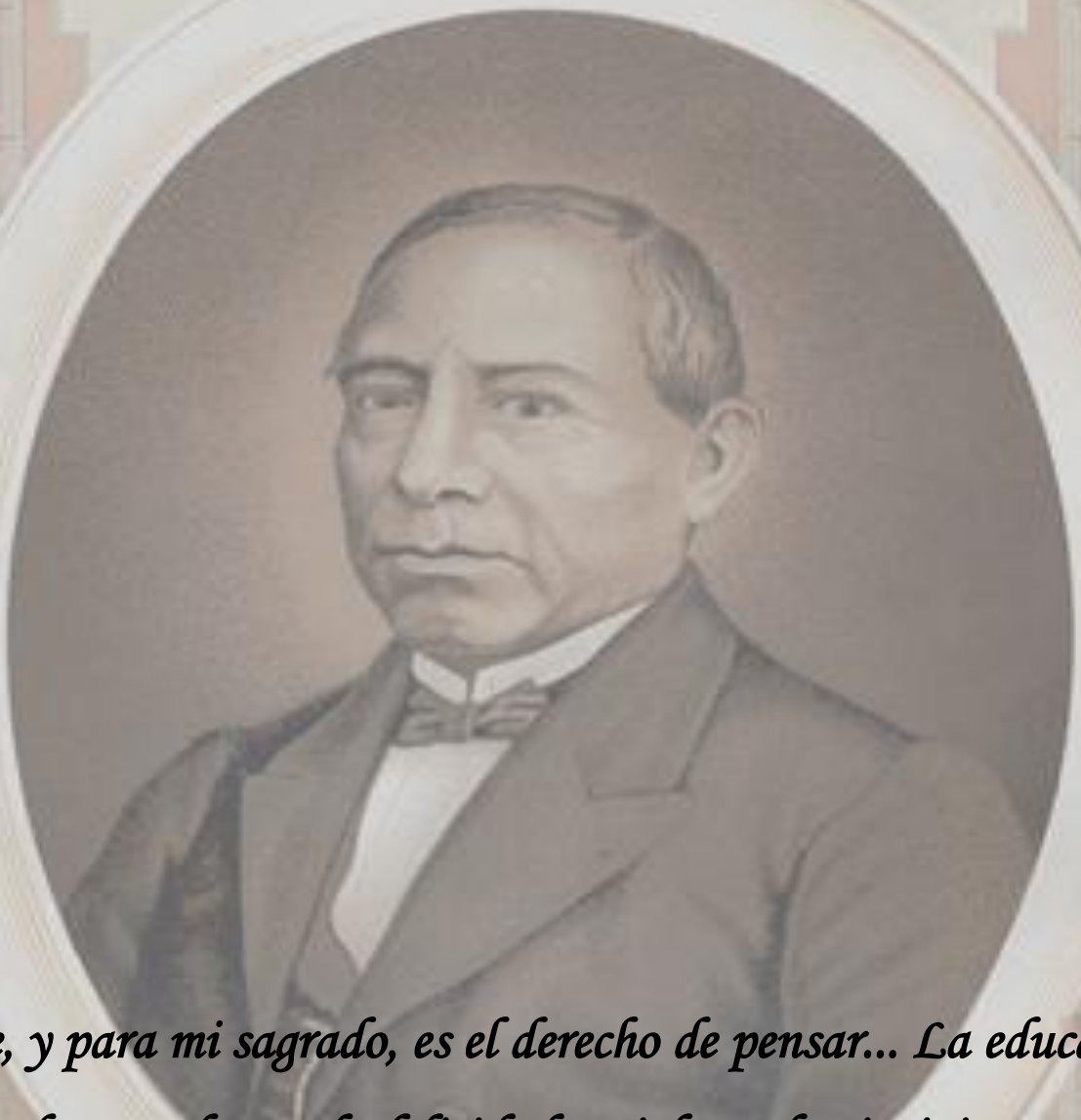
Título: *"Análisis y Diseño del Modelo de Factibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas".*

Autor(es): Juan Carlos Hernández Rabelo.
Maidel Díaz Paredes.

Tutor: Ing. Marieta Peña Abreu.

Ciudad de La Habana

Año 2010



Libre, y para mi sagrado, es el derecho de pensar... La educación es fundamental para la felicidad social; es el principio en el que descansan la libertad y el engrandecimiento de los pueblos.

D. BENITO JUÁREZ

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Juan Carlos Hernández Rabelo

_____

Firma del Autor

Maidel Díaz Paredes

_____

Firma del Autor

Marieta Peña Abreu

_____

Firma del Tutor

DEDICATORIA

A nuestro Comandante en Jefe Fidel Castro Ruz y a nuestra revolución, que aportó, apoyó y siguió constantemente la brillante idea de fundar la Universidad de las Ciencias Informáticas y que nos permitió cursar estos maravillosos años de estudio.

De Juan Carlos:

A todos los que nunca perdieron la fe en mí.

A mis familiares y amigos.

A todo el que en algún momento me extendió su mano y a los que no también.

A todos, gracias.

De Maidel:

A todo el que confió en mí.

A mi familia y amigos.

A todo aquel que me brindó apoyo en mi vida.

A los que me dieron la espalda porque también me dieron fuerzas.

A todos, gracias.

AGRADECIMIENTOS

De Juan Carlos:

A mis familiares y amigos que estuvieron presentes en todo momento a lo largo de toda mi vida de estudiante brindando el apoyo necesario en el momento justo.

A mi abuela Librada especialmente, por ser tan comprensiva conmigo, por ser mi bastón, mi gran apoyo, mi razón de ser, te quiero mimi.

A mi mamá, por todo el amor, por su confiar ciegamente, por todos los sacrificios que has hecho por mi, gracias mami por traerme a este mundo, te adoro.

A mi padre, por todos los consejos, por esa mano siempre extendida, por tu prestigio y por regalarme tu carácter y tu fuerza el día que viene al mundo, te quiero padre.

A mi tío Juan Antonio, por todo su esfuerzo, apoyo incondicional y constancia para verme llegar aquí donde estoy hoy, eres lo máximo tiito.

A mi tía Gladys, por ser mi defensora, por su gran apoyo y llenar mi vida de gratos momentos, te quiero mucho tía.

A mi abuelo, mis tíos, mi hermana, a mis primos, a todos gracias.

A mis vecinos, por ser parte de mi familia y hacer al igual que mi familia mis problemas suyos y estar siempre pendientes de mí.

A todos mis amigos, que fueron mi fuerza y mi apoyo en estos años de estudio.

A José Ángel y su familia, por prácticamente convertirse en mi segunda familia, los quiero de corazón.

A los profesores sin excepción de ninguno, que durante estos años han hecho de mi una persona más capacitada y preparada para enfrentar los grandes retos que han de venir en lo adelante como todo un profesional.

A nuestra tutora y a todos los que de una forma y otra hicieron posible la realización de nuestra tesis.

AGRADECIMIENTOS

De Maidel:

A mi madre del alma por ser tan buena, comprensiva y paciente todos estos años conmigo, muestra de sacrificio y empeño para cualquier ser humano. Sus consejos y reflexiones siempre estarán conmigo. Es lo que más quiero en el mundo, tú lo sabes mami.

A mi hermano, el mejor regalo y amigo que me dio la vida, que siempre me ha guiado por el camino correcto y me ha enseñado muchas cosas de las cuales estoy muy orgulloso. Te quiero mucho hermanito.

A mi padre por estar ahí en los momentos difíciles que me ha deparado el destino y enfrentarlos como si fueran de él, por ayudarme siempre y quererme mucho, gracias viejo.

A mis abuelos aunque ya muchos no están, en especial a mi abuelita mamá por quererme muchísimo y estar ahí a mi lado cuando más falta me ha hecho siempre, te quiero con la vida mamá.

A mi sobrinito Maikol por traerle mucha alegría a la familia, además por ser el nieto fundador, que lo quiero con la vida.

A Dayanna por compartir mis alegrías y tristezas durante mucho tiempo, por dedicarme todo su tiempo, su entrega y dedicación, por calar tan hondo en mi alma, te quiero mucho mucho negrita.

A mis tías, tíos y primos que siempre me han ayudado mucho para llegar a ser lo que siempre desde pequeño soñé.

A todos mis profesores que desde pequeño han ido dejando una huella profunda en mi, para llegar a ser el profesional que siempre soñé ser.

A mis amigos de la universidad y de Sancti Spíritus por compartir muchos momentos buenos y malos conmigo, que nunca los olvidaré esté donde esté, siempre serán mis amigos.

RESUMEN

En el curso 2008-2009 como parte de un trabajo de diploma para optar por el título de ingeniero en ciencias informáticas, se hizo una investigación comprobándose, que los métodos de evaluación para la selección de proyectos eran insuficientes y estaban en constante perfeccionamiento, ocasionando muchas veces que se cometieran errores a la hora de determinar la creación de proyectos. Para dar solución a dicha situación se propuso el Modelo de Factibilidad para la Evaluación de Proyectos de Software, obteniéndose buenos resultados en cuanto a la estimación de factibilidad de los proyectos a partir de su aplicación, hay que señalar que dichos procesos de estimación se basan en la aplicación de métodos de evaluación de proyectos y de criterios de evaluación que deben ser definidos manualmente, haciéndose un tanto engorrosa la aplicación del mismo, por lo que el modelo aún no se ha generalizado hacia todos los proyectos de la universidad, reflejándose la necesidad de que sea implementado.

En el presente trabajo se desarrolla el análisis y diseño del Modelo de Factibilidad para la Evaluación de Proyectos de Software, lo que posibilitará la implementación del sistema. Se generaron los artefactos: Modelo de dominio, Especificación de requisitos de software, Especificación de casos de uso del sistema, Prototipo de interfaz no funcional del sistema, Diagramas de clases del diseño y Diagramas de secuencia. Para lo que se utiliza el Proceso Unificado de Desarrollo de Software (RUP) como metodología de desarrollo de software, Visual Paradigm como herramienta CASE, UML como lenguaje de modelado y Java como lenguaje de programación.

A partir de esta propuesta y con su seguimiento, se espera obtener un modelo informatizado que sea de fácil uso y se aplique en todos los proyectos de la Universidad.

PALABRAS CLAVES

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

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La Gestión de Proyectos da sus primeros pasos en la década del 50, a partir de entonces ha ido evolucionando y aún continúa haciéndolo, convirtiéndose en la actualidad en un área del conocimiento interdisciplinaria donde las técnicas de dirección, de gestión de recursos humanos y los conocimientos técnicos del área específica de aplicación se funden en un sistema único de trabajo, dirigido por un líder. Las nuevas estructuras organizacionales de los proyectos y la necesidad del perfeccionamiento en los procesos de producción, el entorno empresarial, el mundo de la comercialización y los negocios, se vuelven cada día más competitivos, exigen calidad en los procesos y reclaman tecnología de avanzada. (1)

De ahí que sea de gran importancia reconocer de antemano la viabilidad de un proyecto así como si el resto de las condiciones de ejecución son razonables y si es aconsejable que dicha idea se convierta en un proyecto. Es imprescindible anticipar las dificultades previsibles y consiguientemente poder poner en juego los medios necesarios y las soluciones técnicas a fin de que la ejecución de la obra en cuestión se realice en las debidas condiciones de tiempo, calidad y costo. La decisión de acometer un proyecto entraña siempre un riesgo y supone la adjudicación de recursos importantes, por lo cual no es sensato lanzarse a la ejecución del proyecto sin antes realizar los estudios pertinentes que confirmen la viabilidad y factibilidad del mismo. Algunos de los factores que se deben revisar son: estudios de mercado, estudios tecnológicos, económicos, los recursos humanos, así como los costes de tiempo, entre muchos otros que no se deben descuidar al iniciar un proyecto. (2)

Dentro de la Gestión de Proyectos se puede identificar una importante área encargada de dar respuesta a la interrogante de si es o no factible la realización de un proyecto, este es el caso de la Evaluación de Proyectos. Una de las vías para la aprobación de la realización o no de un proyecto sería determinar las posibilidades de éxito que este tendrá y su posible impacto. La evaluación de proyectos consiste en la recopilación de información pertinente con el fin de facilitar el proceso de selección de proyectos y determinar el valor de cada proyecto.

Cuba a pesar de ser un país subdesarrollado, lucha incansablemente por estar a la altura de los avances de la ciencia y la tecnología a nivel mundial. Con el fin de lograr esta meta ha puesto su empeño y centrado esfuerzos en la creación de espacios donde se practica la inteligencia en bien de la humanidad, una muestra de ello lo constituye la Universidad de las Ciencias Informáticas (UCI), centro que fomenta el desarrollo informático y la creación de software, permitiendo informatizar la sociedad y siendo también vía importante de ingresos a la economía de nuestro país.

La UCI se presenta en el desarrollo de la industria de software como una institución joven, con una comunidad de trabajadores inexpertos en el tema, lo que trae consigo la ejecución de algunos procesos de manera ineficiente. Los métodos de evaluación para la selección de proyectos son insuficientes y están en constante perfeccionamiento, ocasionando, muchas veces, que se cometan errores a la hora de determinar la creación de proyectos. Temas como la gestión de riesgos y la factibilidad técnica en general no se tratan adecuadamente. Los estándares internacionales no se encuentran completamente implantados y generalizados. Los modelos de evaluación existentes no se ajustan al modelo de producción de la UCI convergiendo la mayoría a temas de factibilidad económica. La universidad no dispone de procesos suficientes para la evaluación dentro de la gestión de proyectos de software, lo que provoca la inadecuada selección de proyectos productivos. (3)

Teniendo en cuenta este problema en el curso 2008-2009, tras un estudio de todos los factores, se hizo la propuesta de un Modelo de Factibilidad para la Evaluación de Proyectos de Software en la UCI, sentando las bases de lo que pudiera ser una nueva herramienta en el proceso de selección de las nuevas solicitudes de proyectos que se presentarán en los años venideros, permitiendo mejoras en la evaluación, selección y contribuyendo a la factibilidad técnica y económica en la universidad. Actualmente el modelo es aplicado en algunos proyectos productivos de la Universidad, ejemplo de ello es el conjunto de proyectos que conforman el Centro de Tecnologías de Almacenamiento Análisis de Datos (CENTALAD), obteniéndose buenos resultados en cuanto a la estimación de factibilidad y de viabilidad de los mismos. A pesar de lo antes mencionado, hay que señalar que resulta complejo gestionar todo el conjunto de métodos y técnicas de evaluación de proyectos que emplea para poder estimar cada uno de los aspectos que influyen en la selección de proyectos viables, pues se realizan de forma manual, por lo que no se ha generalizado a todos los proyectos productivos de la universidad. Por lo que se hace necesario su informatización, para ello se debe llevar a cabo un estudio previo para delimitar las funcionalidades que debe contener el sistema, proporcionando así un mejor entendimiento de los requisitos del software a los futuros desarrolladores.

Por lo que surge como **Problema Científico:** ¿Cómo lograr un entendimiento común entre clientes y desarrolladores, que facilite la implementación del Modelo de Factibilidad para la Evaluación de Proyectos de Software? Donde se tiene como **Objeto de Estudio:** Ingeniería de Software, centrando el desarrollo en el **Campo de Acción**: Análisis y Diseño del Modelo de Factibilidad para la Evaluación de Proyectos de Software.

Para dar solución al problema planteado se define como **Objetivo General**: Realizar el análisis y diseño del Modelo de Factibilidad para la Evaluación de Proyectos de

Software, facilitando un entendimiento común entre clientes y desarrolladores y permitiendo una posterior implementación.

Para dar solución al objetivo planteado se trazan las siguientes **Tareas de la Investigación:**

- Elaboración del marco teórico-referencial de la investigación estudiando para ello un conjunto de aspectos importantes tales como, la ingeniería de requisitos y sus principales actividades, el diseño con los procesos y patrones que propone, las principales metodologías de desarrollo del software, las herramientas CASE, los lenguaje de modelado y de programación, así como los principales patrones tanto de casos de uso como del diseño.
- Identificación de los conceptos que se tratan en el Modelo de Factibilidad para la Evaluación de Proyectos de Software.
- Identificación, análisis y especificación de los requisitos de software, con el objetivo de tener todas las funcionalidades requeridas y obtener una definición precisa, completa y verificable.
- Obtención de artefactos del diseño con el objetivo de modelar procesos que deben ser implementados.
- Validación de los resultados obtenidos, garantizando una propuesta que cumpla con las funcionalidades previstas.

Deduciéndose la **Hipótesis** de que: si se realiza el análisis y diseño del Modelo de Factibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas se logrará un entendimiento entre clientes y desarrolladores posibilitando su posterior implementación.

Métodos y técnicas de investigación a utilizar

Para validar metodológicamente la investigación utilizamos 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 hipotético-deductivo: para la definición de la hipótesis de la investigación.
- Sistémico: se plantea el problema y su solución como un todo, realizando un estudio de cada uno de los componentes de la evaluación de proyectos de software, estableciendo dependencias y conexiones entre cada una de las fases para poder lograr un resultado integral e instaurar así un modelo sostenible.

- Método de la modelación: para el diseño es necesario la elaboración de diagramas, figuras y otros artefactos importantes, por lo que se hará uso del método de modelación, pues mediante este se pueden crear abstracciones con el propósito de explicar la realidad.

Empíricos

- Observación: con el objetivo de analizar y captar deficiencias en el proceso de análisis y diseño.

Estructura de la Tesis

El presente trabajo de diploma consta de tres capítulos.

Capítulo 1. Fundamentación Teórica. Se realiza un estudio sobre algunas de las metodologías para el desarrollo de software, los lenguajes de modelado, herramientas CASE que pueden utilizarse para modelar los artefactos que proponen dichas metodologías, así como los diferentes lenguajes de programación. Se tratan aspectos concernientes a la ingeniería de requisitos y el diseño. Se abordan además los principales patrones de casos uso y de diseño existentes.

Capítulo 2. Solución Propuesta. En este capítulo se realizan las actividades de elicitación, análisis y especificación de los requisitos, obteniendo los requisitos funcionales y no funcionales del modelo. Se obtienen además el diagrama de casos de uso y la especificación de los mismos y se obtienen del diseño los diagramas de clases y de secuencia.

Capítulo 3. Validación de los Resultados. En este capítulo se verifica y valida la solución propuesta, a partir de la aplicación de las métricas: para la calidad de la funcionalidad del Documento de Especificación de Requisitos, evaluación de la funcionalidad del diagrama de casos de uso, también se emplearon otras para la validación del diseño como, cohesión y tamaño de clases, realizando un análisis de los resultados obtenidos con las mismas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

En el presente capítulo se analizan algunos aspectos importantes utilizados para la estimación de factibilidad de los proyectos. Se muestran aspectos concernientes a la ingeniería de requisitos y al diseño. Se exponen los principales patrones existentes para la concepción de los casos de uso (CU) y del diseño. Se hace alusión a algunas de las principales metodologías y herramientas más usadas en el desarrollo de software.

1.2 ESTUDIO DE FACTIBILIDAD EN PROYECTOS DE SOFTWARE

La investigación de factibilidad en un proyecto consiste en descubrir cuáles son los objetivos de la organización, luego determinar si el proyecto es útil para que la institución logre sus objetivos. La búsqueda de estos objetivos debe contemplar los recursos disponibles o aquellos que la institución puede proporcionar, nunca deben definirse recursos que no estén al alcance o no existentes.

El estudio de factibilidad sirve para recopilar datos relevantes sobre el desarrollo de un proyecto y en base a ello tomar la mejor decisión, si procede su estudio, desarrollo o implementación. (4)

Factibilidad se refiere a la disponibilidad de los recursos necesarios para llevar a cabo los objetivos o metas señalados, la factibilidad se apoya en tres aspectos básicos: operativo, técnico y económico. El éxito de un proyecto está determinado por el grado de factibilidad que se presente en cada una de los tres aspectos anteriores. (4)

1.3 TENDENCIAS DE LAS TÉCNICAS USADAS PARA LA EVALUACIÓN DE PROYECTOS

Las técnicas y métodos empleadas para evaluar los diferentes proyectos se pueden clasificar en tres grupos fundamentales, según las características de cada uno de ellos: Los métodos cuantitativos basados fundamentalmente en criterios económicos, los métodos multicriterio basados en aspectos cualitativos que son evaluados por expertos y la revisión por pares, la cual se basa, en el juicio que dan los expertos que trabajan la temática llegando a una conclusión final a través del consenso. (5)

De acuerdo al estudio realizado a cada uno de estos métodos, se ha hecho una selección de ellos agrupándose, según su aplicación, en dos grupos: en la evaluación de los proyectos de

forma individual y la evaluación a partir de una cartera, facilitando así la selección de estos en el proceso de evaluación, consultar Figura 1.

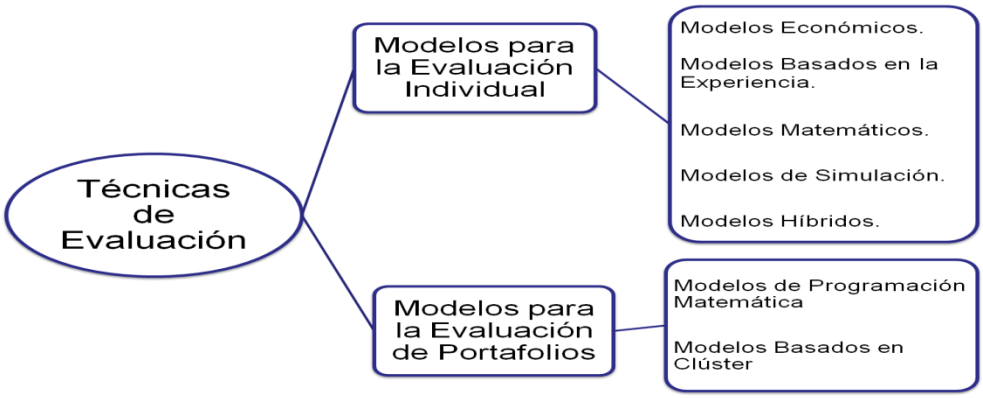


Figura 1 Gráfico de Técnicas de Evaluación de Proyectos.

1.4 MODELO DE FACTIBILIDAD PARA LA EVALUACIÓN DE PROYECTOS DE SOFTWARE

El modelo está estructurado por tres fases fundamentales: Iniciación, Evaluación y Comunicación. Cada una de estas contiene entradas, técnicas y herramientas así como un conjunto de salidas, proceso representado en la Figura 2.

Figura 2 Estructura del Modelo de Evaluación.

La evaluación de los diferentes proyectos se realiza de igual forma organizacional que en un proyecto real, por lo cual es integrado por un grupo de recursos humanos que son los ejecutores de la evaluación, mostrados en la Figura 3.

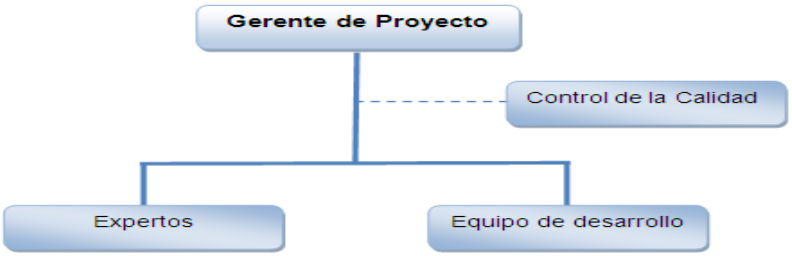


Figura 3 Estructura de del Equipo de Desarrollo.

Además el modelo basa su funcionamiento en un Meta evaluador que permite incluir tantos métodos y criterios de evaluación como se desee siempre que cumplan con todo el Proceso de evaluación, ver Figura 4. (3)

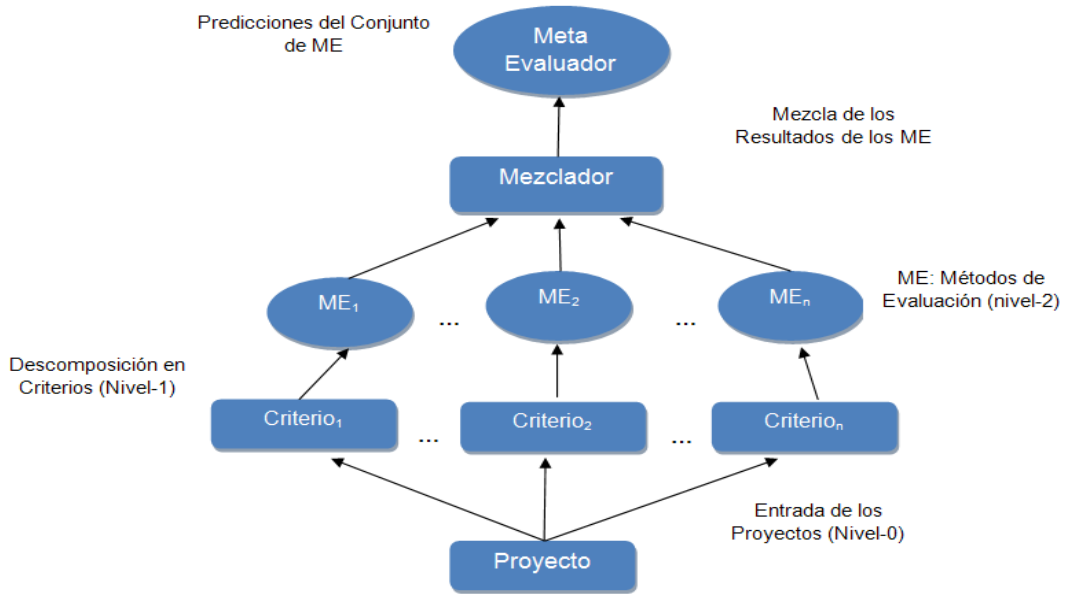


Figura 4 Proceso de Funcionamiento del Meta Evaluador.

1.5 PROCESO DE DESARROLLO DE SOFTWARE

El software según Pressman: “es la máquina que conduce a la toma de decisiones comerciales”. Los desarrolladores capacitados, preparados y responsables de construir el software se conocen como ingenieros del software. (6)

El proceso de desarrollo del software ha estado evolucionando constantemente. A medida que se ha desarrollado la industria informática, los sistemas de software han necesitado ampliarse y ganar en complejidad y calidad puesto que constituyen actualmente un elemento común en la sociedad, convirtiéndose día a día en imprescindibles para la industria, el comercio y las personas. La necesidad de aplicar conceptos, técnicas y métodos ingenieriles para el desarrollo de sistemas de software de alta calidad, que disminuyan los plazos de entrega y de costo ha dado

lugar a la aparición de la Ingeniería de Software (IS), disciplina que surge para aplicar estos principios.

Diferentes autores han definido la IS de diversas maneras.

IS es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software. (7)

Boehm en su definición destaca que la IS es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora, resaltando la necesidad de generar documentación asociada al software, ya sea durante su desarrollo, durante su extensión hacia los usuarios o clientes, o para su posterior mantenimiento. (8)

Para obtener un producto de software con calidad actualmente es imprescindible la aplicación correcta de las técnicas y las buenas prácticas de la ingeniería del software. Esta disciplina desempeña un importante papel pues con la aplicación de sus métodos y técnicas posibilita el control del proceso de desarrollo y aumenta la productividad de los desarrolladores, a los que brinda las bases para construir sistemas de software con calidad y eficiencia.

1.6 INGENIERÍA DE REQUISITOS

La Ingeniería de Requisitos (IR) es un área clave en el desarrollo de software que se enmarca en sus primeras etapas de desarrollo. Cumplir con las necesidades y las expectativas planteadas por los clientes es la razón primera de una práctica sólida del trabajo con requisitos. (9)

Cometer errores en el momento de entender los requisitos es fatal, puesto que estos tienen el potencial para ser los de mayor costo porque muchas decisiones de diseño dependen de ellos.

Para Pressman la “IR ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software”. (6)

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

La IEEE define un requisito como:

- Una condición o capacidad necesaria para un usuario para resolver un problema o alcanzar un objetivo.

- Una condición o capacidad que debe ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación u otro documento formalmente impuesto.
- Una representación documentada de una condición o capacidad dada en los puntos 1 o 2. (7)

Los requisitos se pueden agrupar en dos grandes distinciones como son, los requisitos funcionales y los requisitos no funcionales, que a su vez se dividen en varias categorías.
Requisitos funcionales (RF): definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Además, son independientes de las tecnologías usadas por el producto. (10)
Requisitos no funcionales: son requisitos de software que describen no lo que el software hará, sino como lo hará. Estos son difíciles de verificar/testear, y por ello son evaluados subjetivamente. (11)

1.6.1 Actividades de la Ingeniería de Requisitos

La IR se divide en etapas lógicamente bien definidas, las cuales especifican los pasos, las tareas y las técnicas que se deben emplear para interactuar con los clientes y especificar correctamente el sistema. El buen entendimiento de cada una de las etapas 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 desarrollo del proyecto y una guía para especificar en detalles el sistema solicitado.
Elicitación de Requisitos: es la actividad mediante la que el equipo de desarrollo de un sistema de software extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema. (12) Para dar cumplimiento a la misma se aplican un conjunto de técnicas del levantamiento de requisitos que son: (13)

- **Entrevistas:** resultan una técnica muy aceptada dentro de la IR y su uso está ampliamente extendido. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural. Abarca cuatro pasos: identificación de los entrevistados, preparación de la entrevista, realización de la entrevista y documentación de los resultados. No es fácil de aplicar, se requiere que el entrevistador sea experimentado y tenga capacidad para elegir bien a los entrevistados y obtener de ellos toda la información posible en un período de tiempo siempre limitado. Aquí desempeña un papel fundamental la preparación de la entrevista.
- **Cuestionarios:** requiere que el analista conozca el ámbito del problema en el que está trabajando. Consiste en redactar un documento con preguntas cuyas respuestas sean cortas y concretas, o incluso cerradas por unas cuantas opciones en el propio cuestionario. Este

cuestionario será cumplimentado por el grupo de personas entrevistadas o simplemente para recoger información en forma independiente de una entrevista.

- **Desarrollo conjunto de aplicaciones:** constituye una alternativa a la entrevista individual que ahorra tiempo al evitar que se analice la opinión de cada cliente por separado. Además, la documentación es revisada por todo el grupo presente.
- **Tormenta de ideas:** es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Suele estar formada por un número de cuatro a diez participantes, uno de los cuales es el jefe de la sesión, encargado más de comenzar la sesión que de controlarla. Puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de elicitación, cuando los requisitos son todavía muy difusos.
- **Mapas conceptuales:** son grafos en los que los vértices representan conceptos y las aristas representan posibles relaciones entre dichos conceptos. Estos grafos de relaciones se desarrollan con el usuario y sirven para aclarar los conceptos relacionados con el sistema a desarrollar. Esta es una técnica muy usada en el levantamiento de requisitos, dada la facilidad de entendimiento que provee para los usuarios. Es recomendable que en este sentido el equipo de desarrollo elabore el mapa conceptual basado en el lenguaje del usuario. No obstante, es propicio que se acompañe de una descripción textual pues en casos complejos puede llegar a ser un tanto ambiguo.

Análisis de Requisitos: una vez los requisitos están recopilados, se agrupan por categorías y se organizan en subconjuntos. Se estudia cada requisito en relación con el resto, se examinan los requisitos según su consistencia, completitud y ambigüedad, y se clasifican en base a las necesidades de los clientes/usuarios. (12)

Especificación de Requisitos: una especificación puede ser un documento escrito, un modelo gráfico, un modelo matemático formal, una colección de escenarios de uso, un prototipo o una combinación de lo anterior. A través de la especificación de requisitos se puede negociar concretamente lo que el sistema debe hacer y cumplir para evitar retrasos y costes agregados por la mitigación de errores. Es el punto de partida de la estimación de costo, tiempo y esfuerzo del proyecto y tiene una incidencia marcada en los procesos de gestión de la calidad del producto a entregar. (12)

Validación de Requisitos: la validación de requisitos tiene como misión demostrar que la definición de los requisitos define realmente el sistema que el usuario necesita o el cliente desea. Examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados

hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto. (12)

Administración de Requisitos: la administración de requisitos es un conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar y seguir los requisitos y los cambios en cualquier momento. La administración de los requisitos incluye todas las actividades que mantienen la integridad y exactitud de los mismos a medida que el proyecto progresa. En esta etapa se enfatizan:

- Control de los cambios de los requisitos que están sobre la línea base definida.
- Control de versiones tanto de requisitos individuales como del documento de requisitos. (12)

1.7 METODOLOGÍAS DE DESARROLLO

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software. (14) Surgen a raíz de la necesidad de controlar y documentar proyectos cada vez más complejos. El empleo de las mismas es necesario si se desean gestionar adecuadamente los proyectos, aplicarlas en una organización es una tarea difícil y compleja. El éxito en su utilización depende de múltiples factores, antes de decidirse por una en función de sus características se debe reflexionar acerca de sus aspectos determinantes, así como las ventajas sobre las otras. El término de metodología, se define como un conjunto de métodos eficientes orientados a conseguir un objetivo propuesto. Son un conjunto de procesos que organizados dan una secuencia de pasos a seguir para obtener los hitos propuestos y finalmente el producto final. Como no existe una metodología de software universal, esta debe guiar a la organización en el desarrollo de sus objetivos. Por ello, la organización, su estructura, canales de información, recursos humanos y físicos, deben ser los adecuados para garantizar el correcto funcionamiento de los procesos y métodos definidos. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar, inspirados por otras disciplinas de la ingeniería. En dependencia a lo que se quiera construir y de los componentes de que se disponga, se utilizará una metodología u otra. Sin embargo, las características de cada proyecto (equipo de desarrollo o recursos) exigen que la metodología sea configurable y flexible a sus necesidades. (15)

1.7.1 Metodologías de Desarrollo Tradicionales

Las metodologías de desarrollo tradicionales modelan tres aspectos o dimensiones de los sistemas informáticos: estructura de la información, funciones y comportamiento. Están guiadas por una fuerte planificación durante todo el proceso de desarrollo, donde se realiza una intensa

etapa de análisis y diseño antes de la construcción del sistema, se centran en el control del proceso, estableciendo las actividades involucradas, los artefactos que se deben producir y las herramientas y notaciones que se usarán.

El Proceso Unificado de Desarrollo de Software (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, es un producto de Rational Software Corporation (IBM). (16)

Tiene como objetivo asignar tareas y responsabilidades para producir software de alta calidad, buscando satisfacer las necesidades de los clientes, ajustándose al presupuesto y a los tiempos estimados. RUP ofrece diferentes subprocesos, que brindan un marco de trabajo adaptable y extensible a las necesidades de cada organización. Se caracteriza por ser: (16)

- 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 requisitos.

Los elementos de RUP son:

- **Actividades:** procesos que se llegan a determinar en cada iteración
- **Trabajadores:** personas o entes involucrados en cada proceso
- **Artefactos:** documentos, modelos, o elementos de modelo.

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.

- **Fase de concepción o inicio:** Esta fase tiene por finalidad definir la visión, los objetivos y el alcance del proyecto, tanto desde el punto de vista funcional como del técnico.
- **Fase de elaboración:** Esta fase tiene como principal finalidad completar el análisis de los casos de uso y obtener la línea base de la arquitectura del sistema.
- **Fase de construcción:** Esta fase está compuesta por un ciclo de varias iteraciones, en las cuales se van incorporando sucesivamente los casos de uso, de acuerdo a los factores de

riesgo del proyecto. Este enfoque permite por ejemplo contar en forma temprana con versiones del sistema que satisfacen los principales casos de uso.

- **Fase de transición:** Esta fase se inicia con una primera versión del sistema y culmina con el sistema en fase de producción. (16)

En RUP se han agrupado 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:

- **Modelamiento del negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- **Análisis y diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (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.
- **Prueba (Testeo):** Busca los defectos a los largo del ciclo de vida.
- **Instalación o despliegue:** Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
- **Administración del proyecto:** Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- **Administración de configuración y cambios:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.
- **Ambiente:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

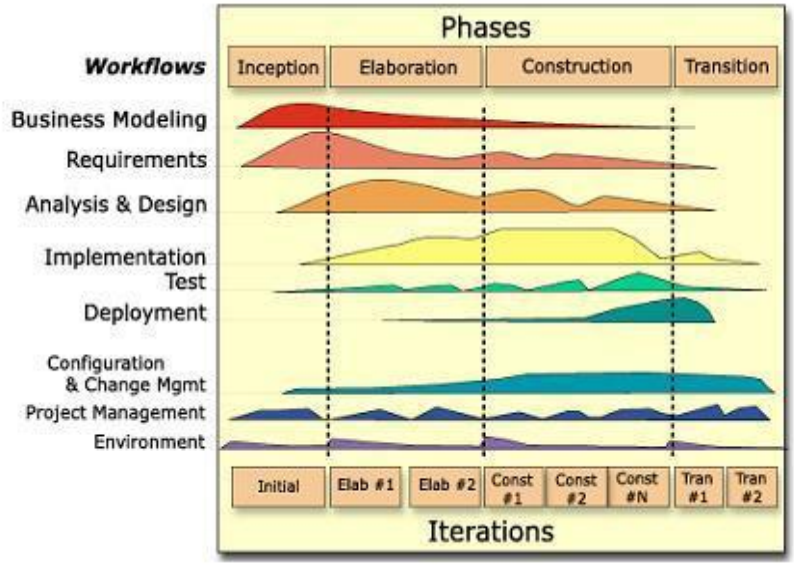


Figura 5 Fases e Iteraciones de la Metodología RUP

Cada una de estas iteraciones debe ser clasificada y ordenada según su prioridad, y cada una se convierte en un grupo de entregables al cliente. Trae como beneficio la retroalimentación en cada iteración.

RUP tiene bien definido sus procesos orientados a objetos, puede ser menos pesado si se es capaz de aceptar y adaptar a las condiciones esperadas de cada proyecto. Además de traer consigo un grupo de beneficios como la estandarización, facilita el entendimiento por parte de los usuarios del software, que no necesariamente deben tener conocimientos previos sobre el tema y facilita además, el desarrollo del producto a distancia de los clientes. (16)

Plataforma de Solución de Microsoft (MSF)

MSF es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. (MSF) Es un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, es una serie de modelos que puede adaptarse a cualquier proyecto de tecnología de información. (17)

Características de MSF: (17)

- *Adaptable:* es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- *Escalable:* puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

- *Flexible:* es utilizada en el ambiente de desarrollo de cualquier cliente.
- *Tecnología Agnóstica:* porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

Los modelos que componen MSF se encargan de planificar las diferentes partes implicadas en el desarrollo de un proyecto, ellos son, Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño del Proceso y Modelo de Aplicación.

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. (18)

Figura 6 Metodología MSF

MSF enfoca el análisis y diseño a través del 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 a través de 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.

Diseño Conceptual: Establece los conceptos que especifican las necesidades de los usuarios. Se debe comenzar a estructurar la solución propuesta, para ello propone la elaboración de los siguientes artefactos:

- 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:

- 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 ello propone la elaboración de los siguientes artefactos:

- 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. (18)

1.7.2 Metodologías de Desarrollo Ágiles

Las metodologías ágiles aparecen como una posible respuesta de solución para algunos proyectos actuales, donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad.

Programación Extrema (XP)

Programación extrema, del inglés Extreme Programming (XP), es una de las conocidas metodologías de desarrollo de software ágiles. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. (19)

Esta metodología ha sido diseñada para solucionar el eterno problema del desarrollo de software por encargo: entregar el resultado que el cliente necesita a tiempo.

Características de XP:

La metodología se basa en**:**

- *Pruebas Unitarias:* Se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándose en algo hacia el futuro, se puedan hacer pruebas de las fallas que pudieran ocurrir. Es como si se adelantara a obtener los posibles errores.
- *Refabricación:* 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

haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

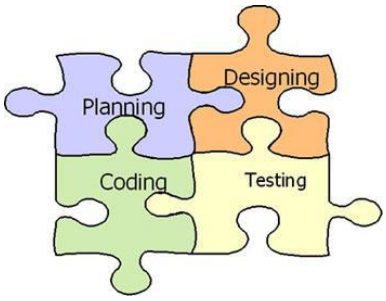


Figura 7 Metodología Extreme Programing

Lo fundamental de XP es:

- **La comunicación**: entre los usuarios y los desarrolladores.
- **La simplicidad**: al desarrollar y codificar los módulos del sistema.
- **La retroalimentación**: concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

Críticas a XP.

- No escalable y con altos riesgos si existen fallas en la arquitectura.
- Altos riesgos si no hay capacidad/estabilidad en las personas.
- La programación de a pares es un intento por solventar la falta de análisis.
- Puede caer en el modelo de codificar y probar.
- Vagas nociones de aseguramiento de la calidad. (19)

Scrum

Proceso ágil que sirve para administrar y controlar el desarrollo de software. El desarrollo se realiza en forma iterativa e incremental. Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. Las iteraciones en general tienen una duración entre dos y cuatro semanas. Está diseñado especialmente para adaptarse a los cambios en los requisitos, estos se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares, de esta forma se puede adaptar en tiempo real el producto que se está construyendo a las necesidades del cliente.

Sus principales características se pueden resumir en dos. La primera es que realiza iteraciones muy cortas, que tienen como resultado un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, destacándose la

reunión diaria del equipo de desarrollo de aproximadamente quince minutos para la coordinación e integración. (14)

1.7.3 Justificación de la metodología seleccionada

Después de realizar el estudio de las principales metodologías de desarrollo de software por parte del equipo de desarrollo se decide aplicar RUP como proceso rector, por adaptarse a las características y necesidades del sistema, hace énfasis en realizar una buena captura de los requisitos, por la necesidad de generar la mayor cantidad de documentación permitiendo perfeccionar el trabajo realizado y una posterior implementación del sistema, asegura la producción de un software de alta calidad que reúna las necesidades de los usuarios finales dentro de un plan y un presupuesto predecible, provee un enfoque disciplinado para asignar tareas y responsabilidades, reduce en gran medida el riesgo que representa la construcción de sistemas, porque evoluciona de forma incremental.

1.8 LENGUAJE DE MODELADO

Hoy en día para el desarrollo de un software es necesario contar con un plan bien organizado. Un cliente tiene que comprender qué es lo que hará un equipo de desarrolladores; además tiene que ser capaz de señalar cambios si no se han captado claramente sus necesidades. Para ello todo el proceso de software debe representarse de manera sencilla, que permita a los usuarios entender su estructura, contenido y organización. La clave está en organizar el proceso de diseño de tal forma que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él.

Jacobson plantea que los objetivos de los mismos son: (16)

- Captar y enumerar exhaustivamente los requisitos y el dominio de conocimiento.
- Pensar en el diseño de un sistema.
- Capturar decisiones del diseño a partir de los requisitos.
- Generar productos aprovechables para el trabajo.
- Organizar, encontrar, filtrar, recuperar, examinar, y corregir la información en grandes sistemas. Explorar económicamente múltiples soluciones.
- Domesticar los sistemas complejos.

Lenguaje Unificado de Modelación (Unified Modelling Language, UML)

Es un lenguaje gráfico para visualizar, especificar, construir, documentar y comunicar los artefactos de un sistema de software. UML permite el desarrollo de distintos tipos de diagramas, cada uno de los cuales representa el sistema a especificar, analizar o diseñar desde distintas

perspectivas. Es utilizado para modelar la información del sistema basado en concepto de objetos. (20)

Las funciones principales de UML son:

- Visualizar: Permite expresar de una forma gráfica un sistema de forma que otro lo pueda entender.
- Especificar: Permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

UML es un lenguaje flexible de modelado que permite definir modelos de análisis y modelos del diseño.

De forma general las principales características son: (20)

- Lenguaje unificado para la modelación de sistemas.
- Soporta tecnología orientada a objetos.
- El cliente participa en todas las etapas del proyecto.
- Corrección de errores viables en todas las etapas.
- Soporta RUP.
- Especifica, mediante los diagramas, cómo se estructura y se comporta el sistema.
- Permite obtener un “plano del sistema”.
- Reduce los costos y el tiempo de desarrollo.
- Representa una colección de las mejores prácticas de ingeniería que tienen una probación exitosa en la modelación de sistemas largos y complejos.

Notación de Modelado de Procesos de Negocio

BPMN es un nuevo estándar de modelado de procesos de negocio, en donde se presentan gráficamente las diferentes etapas del proceso del mismo. La notación ha sido diseñada específicamente para coordinar la secuencia de procesos y los mensajes que fluyen entre los diferentes procesos participantes, incluyendo la unión con el diseño y la implementación.

Con BPMN un diagrama puede ser transformado en un código ejecutable automáticamente, sin la necesidad de programación. De esta forma, el analista de negocios puede definir, diseñar y generar una solución a sus procesos. (21)

Al realizar una secuencia de acciones, BPMN ofrece a los analistas de negocios una forma consistente con su manera de trabajar. Igualmente, sus componentes mapean las dimensiones Qué, Cómo, Cuándo, Dónde y Por Qué.

BPMN presenta como características principales:

- Visibilidad de los procesos de las empresas.
- Mayor flexibilidad y agilidad para adaptación al cambio.
- Brinda la posibilidad de integrar la información del negocio dispersa en diferentes sistemas y permite adquirir una ruta de mejoramiento y eficiencia continua al convertir actividades ineficientes en menores costos a través de uso de tecnología enfocada en procesos.
- Adquirir la habilidad para diseñar, simular y monitorear procesos de manera automática y sin la participación de usuarios técnicos.

Process Modeling Language (PML)

Es un lenguaje de modelado orientado a objetos diseñado para describir el comportamiento del sistema físico mediante estructuras de representación modulares (clases de modelado). Las clases PML representan conceptos físicos que son familiares al modelador. El conocimiento físico declarado por las clases se utiliza para analizar los modelos estructurados, obteniéndose de manera automatizada la representación matemática de las dinámicas de interés. (22)

1.8.1 Justificación de la selección del lenguaje de modelado

Se empleará UML como lenguaje de modelado. Es el lenguaje más utilizado a nivel mundial para modelar los artefactos creados durante el proceso de desarrollo de software en sistemas que no cuenten con procesos de negocio bien definidos. Fue desarrollado conjuntamente con la metodología RUP, responde a todas sus necesidades combinándose para formar una perfecta elección de un equipo de desarrollo que decida usar RUP. Además es un lenguaje del que se tiene gran dominio tanto por el equipo de desarrollo como por el cliente, reduciendo con ello los costes de tiempo que deben emplearse en capacitación para el empleo de otro lenguaje, reúne una colección de las mejores prácticas en la ingeniería que han sido utilizadas con éxito para modelar sistemas, sin importar su envergadura.

1.9 HERRAMIENTAS CASE

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador), son aplicaciones que facilitan el desarrollo de software, reduciendo el esfuerzo, el costo y el tiempo, además de estructurar la documentación asociada a los artefactos generados. (23)

Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o

detección de errores entre otras. Además permiten aumentar las capacidades y habilidades de los analistas, elementos importantes del proceso de desarrollo, pero no las reemplazan.

En el mundo informático se han creado varias herramientas para el desarrollo de la ingeniería de software con el objetivo de desarrollar programas, a continuación se mencionan algunas de las herramientas CASE más utilizadas en la actualidad con sus principales características.

Visual Paradigm

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado ayuda a una construcción más rápida de aplicaciones de calidad, mejores y a un menor coste, además de permitir el dibujo de todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación, así como una serie de tutoriales con demostraciones interactivas y proyectos. (24)

Seguidamente aparecen las razones por las cuales una organización debe elegir VPUML:

- Permite el modelado de base de datos.
- Permite la confección de los prototipos de interfaz.
- Integración perfecta con los principales IDEs que soportan Java, como Microsoft Visual Studio, JBuilder, Eclipse, NetBeans.
- Completa integración con Microsoft Office.
- Herramientas UML más fáciles de usar, ofrece una edición de líneas para diagramas UML, puede crear y especificar un modelo de elementos sin cuadro de diálogos, entre otras funcionalidades.
- Diseño centrado en casos de uso que genera un software de mayor calidad.
- Soporta múltiples plataformas: no importa el Sistema Operativo que este en uso.
- Análisis textual para la identificación de clases candidatas. El análisis textual es una técnica útil y práctica para la captura de los requisitos del sistema y la identificación de las clases candidatas.
- Conversión instantánea de código fuente, archivos ejecutables y binarios en modelos: permite invertir programas de código fuente, archivos ejecutables y binarios y modelos UML de inmediato. Además de idiomas y formatos que incluyen XML, esquema XML, .NET dll o exe, Java de fuente/class/jar, origen de archivo C++ y archivos fuente CORBA IDL.
- Diseño automático de diagramas.

Rational Rose Enterprise Edition 2003

Rational Rose es la herramienta CASE desarrollada por los creadores de UML (Booch, Rumbaugh y Jacobson), 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. (25)

Dentro de las funcionalidades que soporta el Rational Rose están:

- Diagramas UML.
- Desarrollo Orientado al Modelado.
- La generación de código ADA, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables.
- Capacidad de crear definiciones de tipo de documento XML 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

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 ActionScript, C, C++, C# y VB.NET, Java, Visual Basic 6, Python, PHP, XSD, WSDL y otros más. Gestiona la ingeniería de código, normal e inversa, además de una efectiva documentación compatible con Microsoft Word. (26)

Dentro de las funcionalidades que soporta el Enterprise Architect están:

- Diagramas UML.
- CU, Modelos Lógico, Dinámico y Físico.
- Extensiones personalizadas para modelado de procesos.
- Documentación de alta calidad compatible con MS Word.
- Modelado de Datos.
- Ingeniería directa de Base de Datos a DDL e ingeniería inversa de Base de Datos desde ODBC.

- Soporte de pruebas.
- Multi-usuario, con sistema de seguridad.

Se integra a través de plug-ins con herramientas como:

- Eclipse.
- Visual Estudio.Net.

1.9.1 Justificación de la selección de la herramienta CASE

Visual Paradigm, 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, cuenta con un entorno de creación de diagramas para UML, permite la confección de los prototipos de interfaz sin necesidad de hacer uso de una herramienta secundaria y además la universidad cuenta con una licencia de esta herramienta.

1.10 LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. (27)

Dentro de los lenguajes de programación mas usados en el desarrollo de aplicaciones de escritorio se pueden encontrar:

Lenguaje C++

Desde sus inicios, C++ intentó ser un lenguaje que incluyera completamente al lenguaje C (quizá el 99% del código escrito en C es válido en C++) pero al mismo tiempo incorpora muchas características sofisticadas no incluidas en aquel, tales como: programación orientada a objetos, excepciones, sobrecarga de operadores, templates o plantillas. (27)

Lenguaje Java

La sintaxis del lenguaje Java heredó características de C y C++, explícitamente eliminando aquellas que para muchos programadores (según los diseñadores) resultan excesivamente complejas e inseguras. En la actualidad su uso es promovido para el desarrollo de aplicaciones empresariales del lado del servidor, especialmente a través del estándar J2EE, así como en dispositivos móviles (a través del estándar J2ME.) (27)

En realidad, Java hace referencia a un conjunto de tecnologías entre las cuales el lenguaje Java es sólo una de ellas. Por tal motivo muchas veces se habla de la "plataforma Java", la cual es indesligable del lenguaje.

Características principales:

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

Lenguaje C#

El lenguaje C# es una versión avanzada de C y C++ y se ha diseñado especialmente para el entorno .NET. Es un nuevo lenguaje orientado a objetos empleado por programadores de todo el mundo para desarrollar aplicaciones que se ejecuten en la plataforma .NET. C# es un paso muy importante en la evolución de los lenguajes de programación, y es una solución ideal para las aplicaciones de alto nivel. Con C# se puede desarrollar todo tipo de proyectos de aplicaciones cliente/servidor.

C# amplía las capacidades de C, C++, Visual Basic y Java para proporcionar un completo entorno de desarrollo en el que crear aplicaciones. C# mezcla la potencia de C, las capacidades de orientación a objetos de C++ y la interfaz gráfica de Visual Basic. (28)

1.10.1 Justificación del lenguaje de programación seleccionado

Después de realizar un análisis de las características de varios lenguajes de programación se determinó utilizar Java pues es un lenguaje multiplataforma, ofrece gran seguridad ya que cuenta

con ciertas políticas que evitan que se puedan codificar virus con este lenguaje, basa su funcionamiento en la programación orientada a objetos, reduce los costos de desarrollo de software en cuanto a licencias se trata ya que es un software libre, está diseñado para crear software altamente fiable y permite elaborar programas que posibilitan ejecutar varios procesos al mismo tiempo sobre la misma máquina.

1.11 PATRONES DE CASOS DE USO

Los patrones son soluciones simples compuestos por una pareja problema/solución, fundamentadas en la experiencia para problemas específicos y comunes y que se ha demostrado que funcionan y pueden emplearse en diferentes contextos. Algunos de estos son: (29)

Concordancia: toma una subsecuencia de acciones que estén en diferentes partes del flujo de casos de uso y la define por separado.

Extensión Concreta: es de estructura, consiste en dos casos de uso y una relación de extensión. El caso de uso extendido es concreto, es decir, puede ser instanciado por su cuenta como por el caso de uso base. Es aplicable cuando un flujo puede extender el flujo de otro caso de uso, lo que significa que puede ocurrir el proceso del caso de uso base, o puede ocurrir el del caso de uso base con su caso de uso extendido.

Inclusión Concreta: es de estructura. Consiste en dos casos de uso y una relación de inclusión entre el caso de uso base y el caso de uso incluido. Este último puede ser instanciado por sí solo. El caso de uso base puede ser concreto o abstracto. Se utiliza cuando un flujo de datos puede ser incluido en el flujo de otro caso de uso y también puede ejecutarse por sí solo.

CRUD: es de estructura que se basa en la fusión de casos de uso simples para formar una unidad conceptual.

CRUD Completo: es un caso de uso llamado Información de CRUD o Administrar Información, que modela las diferentes operaciones que pueden realizarse en un pedazo de información de un cierto tipo, como crear, leer, actualizar, y eliminar. Debe usarse cuando todos los flujos contribuyen al mismo valor del negocio y estos son cortos y simples.

CRUD Parcial: patrón alternativo que modela uno de las alternativas del caso del uso como un caso de uso separado. Es preferible cuando uno de las alternativas del caso del uso es más significativa, grande, o mucho más compleja que las otras alternativas.

Múltiples actores: captura la concordancia entre actores manteniendo roles separados.

Roles diferentes: consiste en un caso de uso y por lo menos dos actores. Se usa cuando los dos actores juegan papeles diferentes hacia el caso de uso es decir, ellos interactúan de forma diferentemente con el caso del uso.

Roles comunes: puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del caso de uso, solo exista una entidad externa interactuando con cada una de las instancias del caso de uso.

Estos patrones brindan la posibilidad de realizar un mejor y más entendible modelado del sistema. Esto es de gran importancia, pues este modelo es una entrada fundamental para realizar el diseño del sistema.

1.12 DISEÑO DEL SISTEMA

El diseño es uno de los pilares fundamentales en la ingeniería del software. Es una de las actividades técnicas necesarias para la elaboración del software. Entre las tareas fundamentales del diseño están: producir el diseño de datos, diseño arquitectónico, diseño de interfaz y diseño de componentes.

Cuando se diseñan Sistemas Informáticos es indispensable hacerlo de forma correcta. El diseño propuesto tiene que cumplir a cabalidad con los requerimientos del sistema. Es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado… (6); debe ser capaz de facilitar las mejoras del software, tiene que ser entendible por otros profesionales de la especialidad, servir como guía para los demás pasos de la ingeniería de software, permitir la comprobación del sistema fácilmente.

1.12.1 Principios del Diseño

Pressman plantea una serie de principios básicos que se tienen que tener en cuenta en el momento de diseñar, ellos garantizan el correcto funcionamiento del sistema. Dichos principios son: (6)

- En el proceso deben tomarse enfoques alternativos.
- El diseño deberá poderse rastrear hasta el modelo de análisis.
- El diseño no deberá inventar nada que ya esté inventado.
- El diseño deberá minimizar la distancia intelectual entre el software y el problema como si de la misma vida real se tratara.
- El diseño deberá presentar uniformidad e integración.
- El diseño deberá estructurarse para admitir cambios.
- El diseño deberá estructurarse para degradarse poco a poco.
- El diseño no es escribir código y escribir código no es diseñar.
- El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminado.

- El diseño deberá revisarse para minimizar los errores conceptuales.

Cuando estos principios son aplicados correctamente, entonces el diseño creado muestra los factores de calidad internos y externos. Los factores externos son aquellas propiedades del software que se observan fácilmente: velocidad, fiabilidad, grado de corrección, usabilidad. Los internos son importantes para los ingenieros y personal encargo de realizar el proyecto, conducen a un diseño de alta calidad. Para lograrlos es necesario comprender correctamente los conceptos básicos del diseño.

1.12.2 Patrones de Diseño

Los patrones de diseño proponen soluciones exitosas a problemas comunes que se pueden presentar durante el diseño. Un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo desde el análisis hasta el diseño y desde la arquitectura hasta la implementación. Los patrones de diseño tienen como características: (20)

- *Son soluciones concretas:* proponen soluciones a problemas concretos, no son teorías genéricas.
- *Son soluciones técnicas:* indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.
- *Se utilizan en situaciones frecuentes:* ya que se basan en la experiencia acumulada la resolver problemas reiterativos.
- *Favorecen la reutilización de código:* ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar.

Patrones GRASP: los patrones de asignación de responsabilidades (General Responsibility Assignment Software Patterns, GRAPS) describen los principios fundamentales de la asignación de responsabilidades a objetos. Constituyen el fundamento de cómo se va a diseñar el sistema finalmente. Es importante que el diseñador de software domine y aplique estos conocimientos durante la realización de un diagrama de interacción. (20)

Principales patrones GRASP:

Experto:

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

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

Este patrón se usa más que cualquier otro al asignar responsabilidades. El patrón Experto ofrece una analogía con el mundo real.

Creador:

Solución: Asignarle a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos:

- B agrega los objetos de A.
- B contiene los objetos de A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así B es un Experto respecto a la creación de A).

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

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

Bajo acoplamiento:

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

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

El Bajo acoplamiento es un principio que debemos recordar durante las decisiones del diseño: es la meta principal que es preciso tener siempre presente.

Alta cohesión:

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

Alta cohesión es un principio de debemos tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Controlador:

Solución: Asignar la responsabilidad del manejo de mensajes de los eventos del sistema a una clase que represente alguna de las siguientes opciones:

- El sistema global.
- La empresa u organización global.

- Algo activo en el mundo real que pueda participar en la tarea.
- Un manejador artificial de todos los eventos del sistema de un caso de uso (controlador de casos de uso).

Problema: ¿Quién debería encargarse de atender un evento del sistema?

Patrones GOF: estos patrones surgen a raíz del trabajo de un grupo de autores conocido como el Gang of Four (GoF). Ellos recopilaron y documentaron un grupo de patrones de diseño aplicados usualmente por diseñadores de software orientado a objetos.

El grupo GoF agrupó los patrones en tres grandes categorías de acuerdo a su propósito. Este criterio describe la función que el patrón cumple. Los patrones de diseño pueden tener propósito creacional, estructural o de comportamiento: (20)

Patrones Creacionales: Se encargan de las formas de crear instancias de objetos. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

- ***Factory Method:*** centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- ***Abstract Factory:*** permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- ***Builder:*** abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- ***Prototype:*** crea nuevos objetos clonándolos de una instancia ya existente.
- ***Singleton:*** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Patrones Estructurales: Describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

- ***Adapter:*** adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- ***Bridge:*** desacopla una abstracción de su implementación.
- ***Composite:*** permite tratar objetos compuestos como si de uno simple se tratase.
- ***Decorator:*** añade funcionalidad a una clase dinámicamente.
- ***Facade*****:** provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.

- ***Proxy:*** mantiene un representante de un objeto.

Patrones de Comportamiento: Nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

- ***Interpreter:*** define una gramática para un lenguaje dado, así como las herramientas necesarias para interpretarlo.
- ***Chain of Responsibility:*** permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- ***Command:*** encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- ***Iterator:*** permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- ***Mediator:*** define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- ***Memento:*** permite volver a estados anteriores del sistema.
- ***Observer:*** define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado, se notifiquen y actualicen automáticamente todos los objetos que dependen de él.
- ***State:*** permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- ***Visitor:*** permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

1.12.3 Métricas para evaluar el Diseño de Software

Se define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

Definir una métrica específica que proporcione una medida completa de la complejidad del software es bastante difícil. Se han propuesto numerosas métricas, pero todas tienen diferentes puntos de vista, no siempre determinan el mismo valor cuando se mide la complejidad del software. No obstante, su uso es esencial, pues hasta cierto nivel, garantizan un producto con mayor calidad.

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 de 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. (6)

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.

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. (6)

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. (6)

Estas métricas permiten evaluar el diseño, así como saber con anterioridad a qué clase, subsistema y módulo se le debe prestar más atención y designar más recursos para su implementación.

1.13 CONCLUSIONES PARCIALES

En el presente capítulo, luego de profundizar en las principales metodologías y herramientas de desarrollo del software, que posibilitan una mayor productividad, de definir un conjunto de patrones de casos de usos y del diseño, se arriba a las siguientes conclusiones:

- Emplear RUP como metodología de desarrollo, UML como lenguaje de modelado, Visual Paradigm como herramienta CASE y como lenguaje de programación Java, constituyendo estas las bases tecnológicas para el desarrollo de la solución.
- Aplicar las etapas de elicitación, análisis, especificación y validación de los requisitos, desarrollando cada una de las actividades que estas proponen en la ingeniería de requisitos.
- Aplicar patrones de casos de uso, permitiendo la elaboración del diagrama de casos de uso, además permitirán resolver problemas que se puedan presentar a la hora de modelar el sistema de forma rápida y ágil.
- Aplicar los patrones de diseño necesarios para no cometer errores tradicionales en el diseño.
- Utilizar métricas para el chequeo y validación de los artefactos que se generen durante el desarrollo de la solución.

CAPÍTULO 2: SOLUCIÓN PROPUESTA

2.1 INTRODUCCIÓN

En el presente capítulo se explicarán algunos aspectos que ayudarán a una mejor comprensión de las funcionalidades del sistema. Se centra la atención en la obtención de los requisitos funcionales, con el fin de obtener un refinamiento adecuado y los artefactos correspondientes. El diseño consolidará las funcionalidades dispuestas a partir del levantamiento de requisitos, con los diagramas de clases del diseño y los diagramas de secuencia correspondientes a cada escenario y la aplicación de patrones de arquitectura y de diseño como una manera más práctica de describir ciertos aspectos, teniendo en cuenta las cualidades o propiedades con las que debe contar el sistema.

2.2 MODELO DE DOMINIO

La Universidad de las Ciencias Informáticas es un centro que posee una estructura donde los procesos docentes y de producción están interrelacionados, dándosele un gran peso a la producción de software. Los mecanismos de evaluación para la selección de proyectos viables son diversos dependiendo de los intereses de cada área productiva. Dichos mecanismos o técnicas están en constante perfeccionamiento y en ocasiones son insuficiente para determinar si se procede a investigar, desarrollar e implementar un proyecto determinado. La universidad por su parte no ha concebido un estándar por el que se rijan todos los centros productivos para resolver tal problema. Al analizar el entorno relacionado anteriormente, se percibió que los procesos de negocio del Modelo de Factibilidad para la Evaluación de Proyectos de Software no están bien definidos, por lo que se determinó la confección del modelo de dominio.

El propósito fundamental de este modelo es generar una terminología común y sentar las bases del entendimiento del desarrollo del sistema. Dentro de sus características principales están:

- No hay una cronología.
- No se diferencia entre dentro y fuera del sistema.
- Es global, no por caso de uso.
- No es completo: es más bien esquemático, las asociaciones están resumidas.
- Utilizando UML, un modelo de dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación.

Los principales conceptos identificados son:

- **Gerente del proyecto:** dirige todo el proceso de evaluación, realizando un seguimiento del mismo.
- **Expertos:** su número será variable en función de las necesidades del proceso. Son personas expertas en el contenido de los criterios que se evalúan.
- **Equipo de Evaluación:** está compuesto por los grupos de evaluación, llevan a cabo el proceso de Evaluación.
- **Calidad:** controla durante todo el proceso el cumplimiento de la lista de chequeo.
- **Información del Proyecto:** es un documento inicial que se realiza con el objetivo de obtener elementos para una correcta evaluación del proyecto.
- **Información del Entorno del Proyecto:** es un documento que se obtiene a partir del estudio de diferentes áreas con el fin de obtener una visión externa completa del proyecto a evaluar.
- **Listado de Criterios:** son seleccionados a partir de los documentos recogidos acerca del proyecto y su entorno.
- **Métodos de Evaluación:** son estructuras evaluadoras basadas en fórmulas matemáticas.
- **Listado de Proyectos Priorizados:** es un listado de proyectos ordenados a partir del resultado obtenido al aplicar los métodos y criterios de evaluación.
- **Recomendaciones:** a partir de la evaluación realizada se obtienen un grupo de recomendaciones de los proyectos en función de los criterios de evaluación.
- **Acta de Evaluación del Proyecto:** es un documento que es firmado por ambas partes con el objetivo de dar fin al proceso.
- **Proyecto:** es una planificación que consiste en un conjunto de actividades que se encuentran interrelacionadas y coordinadas, además constituye la entrada que dará comienzo al proceso de evaluación.
- **Cartera de Proyectos:** son un conjunto de proyectos.
- **Grupo Evaluador:** Proponen los criterios de evaluación que posteriormente son avalados por el grupo de expertos.

2.2.1 Realización del Modelo de Dominio

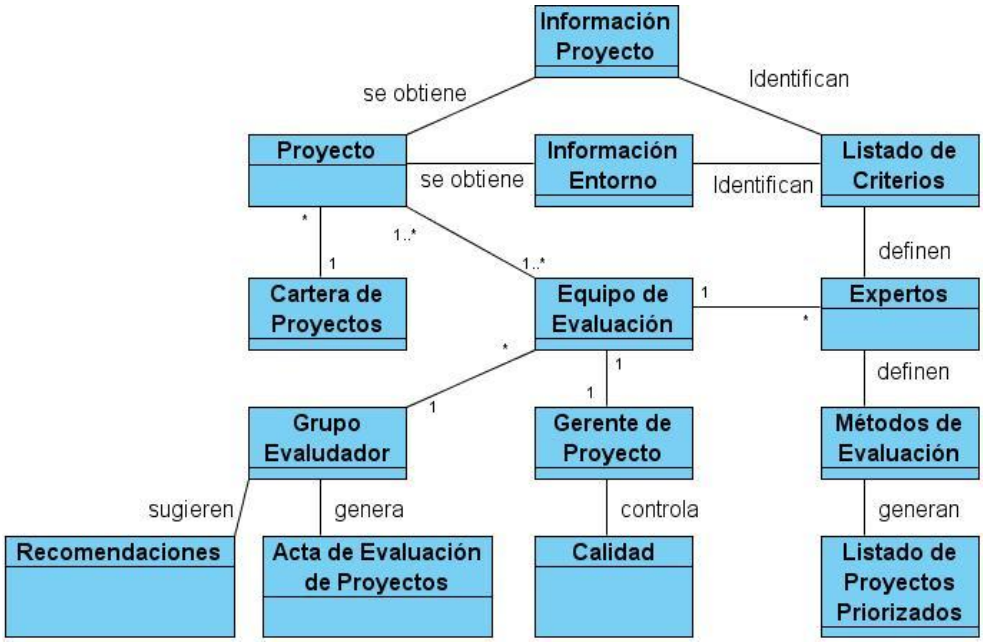


Figura 8 Modelo de Dominio

2.3 ESPECIFICACIÓN DE LOS REQUISITOS DEL SOFTWARE

El propósito general de la captura de requisitos es obtener una descripción correcta de lo que debe de hacer el sistema y delimitar su alcance. Los requisitos juegan un papel importante durante el ciclo de vida de un proyecto. Durante la fase de inicio se identifican la mayoría de los casos de uso para delimitar el sistema y detallar los más importantes.

Para la obtención de los mismos se aplicaron algunas técnicas levantamiento de requisitos. Inicialmente y tratando de lograr una comprensión de las necesidades del cliente se aplicaron las tormentas de ideas, donde cada cliente expuso sus necesidades y emitió su criterio, posteriormente y tratando de reflejar toda la información recopilada en funcionalidades o requisitos, se confeccionaron los cuestionarios a partir de la visión obtenida por el equipo de desarrollo en la primera etapa, dichos cuestionarios se emplearon posteriormente en las entrevistas individuales y finalmente se obtuvo un mapa conceptual donde se reflejaron un grupo de aspectos importantes logrando un entendimiento común entre los clientes y el equipo de desarrollo.

Luego de ser capturados se enumeran a través de requisitos funcionales y no funcionales, todas las acciones que el sistema deberá ser capaz de realizar.

2.3.1 Requisitos Funcionales

Los requisitos funcionales definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Son capacidades o condiciones que el sistema debe cumplir.

Los requisitos funcionales con los que cuenta el Modelo de Factibilidad para la Evaluación de Proyectos de Software se enumeran a continuación y su especificación puede ser consultada en el "Documento Especificación de Requisitos".

RF.1 Permitir autenticación en el sistema.

El sistema permitirá la autenticación de los usuarios en el mismo.

- Capturar usuario y contraseña.
- Validar datos de entrada.

RF.2 Adicionar un Proyecto.

El sistema permitirá que se introduzca un nuevo proyecto.

- Nombre.
- Descripción.

RF.3 Eliminar un Proyecto.

El sistema permitirá que se elimine un proyecto que ya exista.

RF.4 Adicionar Información del Proyecto.

El sistema permitirá que se inserte nueva información del proyecto.

- Área de Datos Identificativos del Proyecto.
- Área de Organización.
- Área Técnica.
- Área de Fundamentación del Proyecto.
- Área de Gestión de tiempo.
- Área de Gestión de Riesgos.
- Área Financiera.

RF.5 Modificar Información del Proyecto.

El sistema permitirá que se modifique información del proyecto existente.

- Área de Datos Identificativos del Proyecto.
- Área de Organización.
- Área Técnica.
- Área de Fundamentación del Proyecto.
- Área de Gestión de tiempo.
- Área de Gestión de Riesgos.
- Área Financiera.

RF.6 Eliminar Información del Proyecto.

El sistema permitirá que se elimine información del proyecto existente.

RF.7 Mostrar Información del Proyecto.

El sistema permitirá que se muestre información del proyecto existente.

RF.8 Registrar la Información del Entorno del Proyecto.

El sistema permitirá que se registre la información del entorno del proyecto.

- Área de Entorno.
- Área de Mercado.

RF.9 Modificar Información del Entorno del Proyecto.

El sistema permitirá que se modifique la información del entorno del proyecto.

- Área de Entorno.
- Área de Mercado.

RF.10 Eliminar la Información del Entorno del Proyecto.

El sistema permitirá que se elimine la información del entorno del proyecto.

RF.11Mostrar la Información del Entorno del Proyecto.

El sistema permitirá que se muestre la información del entorno del proyecto.

RF.12 Adicionar Criterios de Evaluación.

El sistema permitirá que se adicionen nuevos criterios de evaluación.

- Nombre.
- Aplicación.
- Descripción.

RF.13 Modificar Criterios de Evaluación.

El sistema permitirá que se modifiquen criterios de evaluación existentes.

- Nombre.
- Aplicación.
- Descripción.

RF.14 Eliminar Criterios de Evaluación.

El sistema permitirá que se eliminen criterios de evaluación existentes.

RF.15 Mostrar Criterios de Evaluación.

El sistema permitirá que se muestren criterios de evaluación existentes.

RF.16 Ponderar Criterios de Evaluación.

El sistema permitirá que se puedan ponderar los criterios de evaluación con los cuales se trabaja.

RF.17 Adicionar Usuario.

El sistema permitirá que se asignen Usuarios.

- Número del solapín**.**
- Usuario.
- Contraseña.
- Confirmar Contraseña.
- Área.
- Rol.

RF.18 Modificar Usuario.

El sistema permitirá que se modifiquen Usuarios existentes.

- Número del solapín.
- Usuario.
- Contraseña.
- Confirmar Contraseña.
- Área.
- Rol.

RF.19 Eliminar Usuario.

El sistema permitirá que se eliminen Usuarios existentes.

RF.20 Asignar responsabilidades por roles.

El sistema permitirá que se asignen responsabilidades por roles.

RF.21 Adicionar Métodos de Evaluación.

El sistema permitirá que se adicionen métodos de evaluación.

- Nombre.
- Aplicación.
- Descripción.

RF.22 Eliminar Métodos de Evaluación.

El sistema permitirá que se eliminen métodos de evaluación existentes.

RF.23 Mostrar Proyectos Priorizados.

El sistema permitirá que se muestre un listado de proyectos priorizados con documentación de respaldo.

RF.24 Generar Acta de Evaluación de Proyectos.

El sistema permitirá que se cree el acta de evaluación recogiendo en ella si es factible o no, la realización del proyecto.

RF.25 Adicionar Recomendaciones.

El sistema permitirá que se adicionen las recomendaciones recogiendo en ellas, datos que permitirán un mejor desarrollo de los proyectos en un futuro en caso de que así sea.

RF.26 Modificar Recomendaciones.

El sistema permitirá que se modifiquen las recomendaciones, para que así se pueda añadir información importante a las mismas o eliminar en caso de ser necesario.

RF.27 Seleccionar los Expertos.

El sistema permitirá que se seleccionen personas que tienen grandes conocimientos sobre el entorno en el que la organización desarrolla su labor.

RF.28 Obtener competencias de expertos.

El sistema permitirá que se obtenga competencia de los expertos seleccionados para conocer el nivel de conocimiento de los mismos.

RF.29 Adicionar reglas para la selección de criterios.

El sistema permitirá que se establezcan las reglas entre los expertos.

- Conocer el objetivo que persigue cada criterio.
- Los expertos deben permanecer en anonimato.

RF.30 Adicionar estructura del metaevaluador.

El sistema permitirá que se estructure en capas o paralelo, donde todos los evaluadores (métodos de evaluación) son invocados y sus decisiones son seleccionadas o ponderadas de acuerdo con algún criterio de combinación.

RF.31 Adicionar proyectos a evaluar en el nivel 0.

El sistema permitirá que se añada el proyecto a evaluar con el metaevaluador.

RF.32 Adicionar criterios de evaluación al nivel 1.

El sistema permitirá que se adicionen los criterios que resultaron en el proceso de selección de estos.

RF.33 Adicionar métodos de evaluación al nivel 2.

El sistema permitirá que se seleccionen los métodos a utilizar con el metaevaluador.

RF.34 Obtener resultados.

El sistema permitirá que se mezclen los resultados obtenidos para así poder emitir un resultado de acuerdo al proceso de evaluación.

2.3.2 Requisitos no Funcionales

Los requisitos no funcionales especifican criterios que pueden usarse para juzgar las operaciones de un sistema. Se refieren a todos los requisitos que ni describen información a guardar, ni funciones a realizar.

Los requisitos no funcionales con los que cuenta el Modelo de Factibilidad para la Evaluación de Proyectos de Software pueden ser consultados en el "Documento Especificación de Requisitos No Funcionales".

Usabilidad.

El sistema:

- Será usable por cualquier usuario que desempeñe un rol dentro del sistema.
- Se mostrará la información de forma lógica y correctamente estructurada.
- El sistema debe ser fácil de instalar y debe mostrar el avance de la instalación para guiar al usuario del tiempo de instalación que necesita y el que se va ejecutando.

Apariencia o Interfaz Externa.

- El sistema tendrá un menú de acciones para que el usuario pueda disponer de él en la medida de sus necesidades, visible todo el tiempo para propiciar el fácil acceso a las funcionalidades del software.
- El sistema tendrá un ambiente agradable y sencillo, combinado con colores y una estructura amigable que permita que el usuario se adapte con facilidad.
- El sistema permitirá claridad en los servicios que brinda, con términos asociados a los procesos de evaluación de proyectos de software propiciando que el usuario tenga un buen entendimiento de las utilidades que brinda.

Rendimiento.

- El sistema debe garantizar un tiempo de respuesta de 3 a 5 segundos en dependencia de la complejidad de la funcionalidad seleccionada.

Seguridad.

- Se aplicarán las reglas de la "programación segura", mediante el tratamiento de excepciones.
- El sistema permitirá además la verificación sobre acciones irreversibles; es decir, se le solicitará al usuario la confirmación al realizar operaciones como la eliminación.
- El uso y manejo del sistema estará controlado. Toda la información podrá ser consultada solamente por el personal autorizado de acuerdo al rol del mismo.

Portabilidad.

- El sistema debe ser multiplataforma, siendo compatible con cualquier tipo de sistema operativo (SO GNU/Linux, SO Microsoft Windows).

Ayuda.

- El sistema debe contar con un manual de usuario.

- El sistema debe garantizar que la opción de Ayuda esté visible todo el tiempo, para que el usuario la pueda consultar la cantidad de veces necesarias y facilite su trabajo con cada funcionalidad.

Disponibilidad.

- Los usuarios del sistema deben tener acceso (según sus permisos) en todo momento a la información solicitada.

Soporte.

- Es preciso disponer de una documentación apropiada del Sistema para agilizar su Mantenimiento y Configuración.
- Se deben ofrecer servicios de adiestramiento al personal que va a trabajar con el software.

Hardware.

- Se necesitará 1 Gb de espacio libre en disco para la instalación y correcto funcionamiento de la aplicación.
- Se utilizará como procesador Intel Pentium VI con CPU 1.80 GHz.
- Se utilizará una memoria RAM de 512 Mb.
- Se utilizará como capacidad de almacenamiento un disco duro de 40 Gb.

Software.

- Se utilizará como lenguaje de programación Java.
- Se utilizará como Sistema Operativo, Windows 2000/XP/Vista, Linux.
- Se utilizará Máquina Virtual de Java: versión 1.6.
- Se utilizará para el modelado la herramienta Visual Paradigm for UML.
- Se utilizarán los IDE Eclipse 3.2 y NetBeans 6.x.

2.4 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 hemos identificado los actores del sistema, tenemos identificado el entorno externo del sistema.

La definición de los actores del sistema, la descripción de cada uno de ellos, así como el diagrama de actores del sistema, puede ser consultada en el “Documento Modelo de Sistema”.

A continuación se presentan los actores del sistema los actores del sistema y una descripción de los mismos.

Actor	Descripción
Gerente de Proyecto	El actor “Gerente de Proyecto” es un usuario del sistema que dirige todo el

	proceso de evaluación en sus tres fases, selecciona el grupo de evaluadores. Realiza un seguimiento de todo el proceso verificando que todo quede con la calidad requerida. Firma los documentos que se emiten y establece las relaciones de negociación.
Experto	El actor **"**Experto" es un usuario del sistema que tiene permisos para gestionar (crear, modificar y eliminar) los criterios empleados por el sistema para la evaluación de los proyectos de software.
Evaluador	El actor "Evaluador" Lleva a cabo el proceso de Evaluación. Propone los criterios de evaluación que posteriormente son avalados por el grupo de expertos. Emite todos los documentos en el proceso de evaluación.
Administrador	El actor **"**Administrador" es el encargado de administrar, configurar y dar mantenimiento al sistema, además tiene permisos para gestionar los roles.
Usuario	El actor **"**Usuario" es el que realiza la autenticación en el sistema ya sea Gerente de Proyecto, Experto, Evaluador o Administrador.

Una vez conocidos los actores del sistema se elaboró el diagrama de actores del sistema que a continuación se muestra.

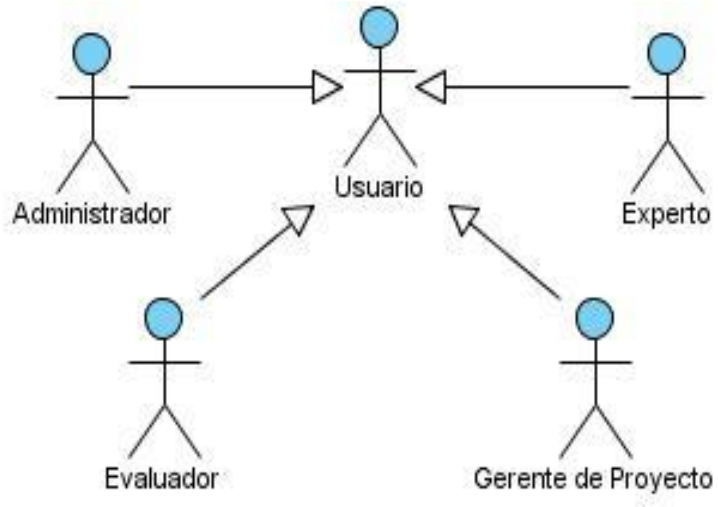


Figura 9 Diagrama de Actores del Sistema

2.5 CASOS DE USO DEL SISTEMA

Los casos de uso son el componente clave del modelado. Su propósito es ilustrar como un sistema permite a un actor cumplir una meta, ilustrando todos los posibles caminos apropiados que ellos pueden tomar para cumplirla, así como las situaciones que podrían hacerlo fallar.

Los casos de uso que cubren las principales tareas o funciones que el sistema ha de realizar son los siguientes:

- Autenticar Usuario.
- Gestionar Proyecto.
- Gestionar Información del Proyecto.
- Gestionar la Información del Entorno del Proyecto.
- Gestionar Criterios de Evaluación.
- Ponderar Criterios de Evaluación.
- Gestionar Usuario.
- Asignar Responsabilidad
- Gestionar Métodos de Evaluación.
- Mostrar Proyectos Priorizados.
- Generar Acta de Evaluación de Proyectos.
- Gestionar Recomendaciones.
- Seleccionar Experto.
- Obtener Proyectos Priorizados.

2.5.1 Diagrama de Casos de Uso del Sistema

Un diagrama de casos de uso es un modelo del sistema que contiene actores, casos de uso y sus relaciones. El diagrama de casos de uso permite que los desarrolladores y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema. El diagrama de casos de uso describe lo que hace el sistema para cada tipo de usuario. Cada usuario se representa mediante uno o más actores.

Luego de identificados los requisitos, 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, para la confección del mismo se emplearon un conjunto de patrones de casos de uso tales como, CRUD Completo para modelar las diferentes operaciones que pueden realizarse en un pedazo de información de un cierto tipo, como crear, leer, actualizar, y eliminar, CRUD Parcial utilizado para modelar las diferentes operaciones que pueden realizarse en un pedazo de información de un cierto tipo, pero en este caso no cumple con todas la operaciones como en el caso anterior, Múltiples actores con roles diferentes permitiendo la concordancia entre actores manteniendo estos roles separados.

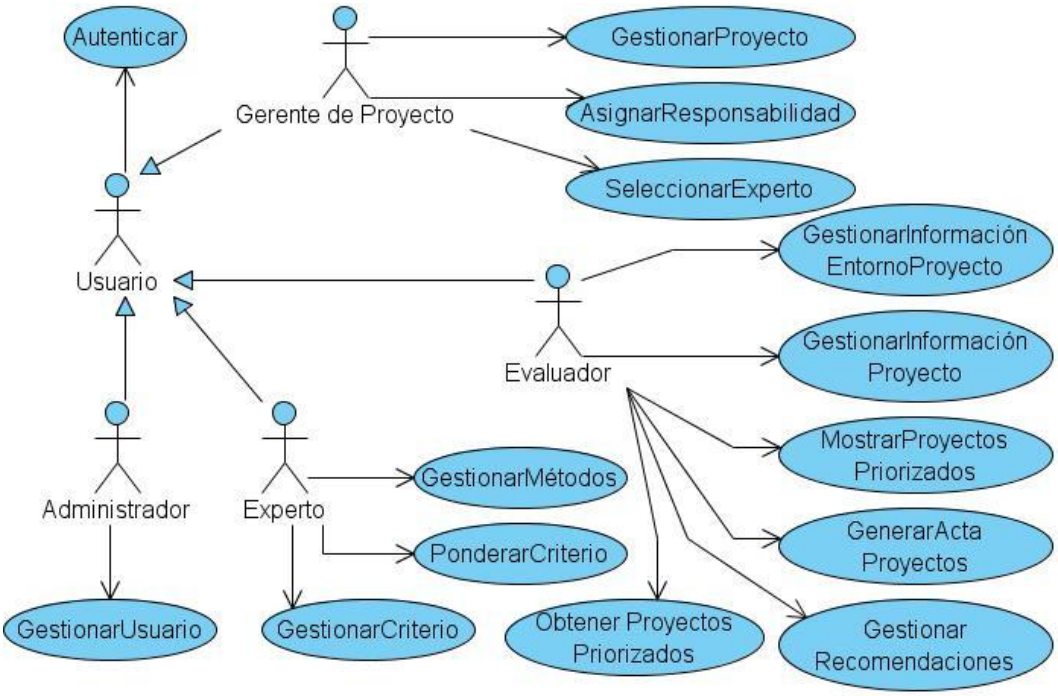


Figura 10 Diagrama de Casos de Uso del Sistema

2.5.2 Especificación textual de los casos de usos

Mediante las descripciones de los casos de uso se especifica la secuencia de eventos que los actores utilizan para completar un proceso a través del sistema. Las descripciones de los casos de uso del sistema así como los prototipos de interfaz podrán ser consultadas en el "Documento Modelo del Sistema".

A continuación se muestra un ejemplo de especificación de casos de uso, correspondiente al caso de uso Gestionar Proyectos:

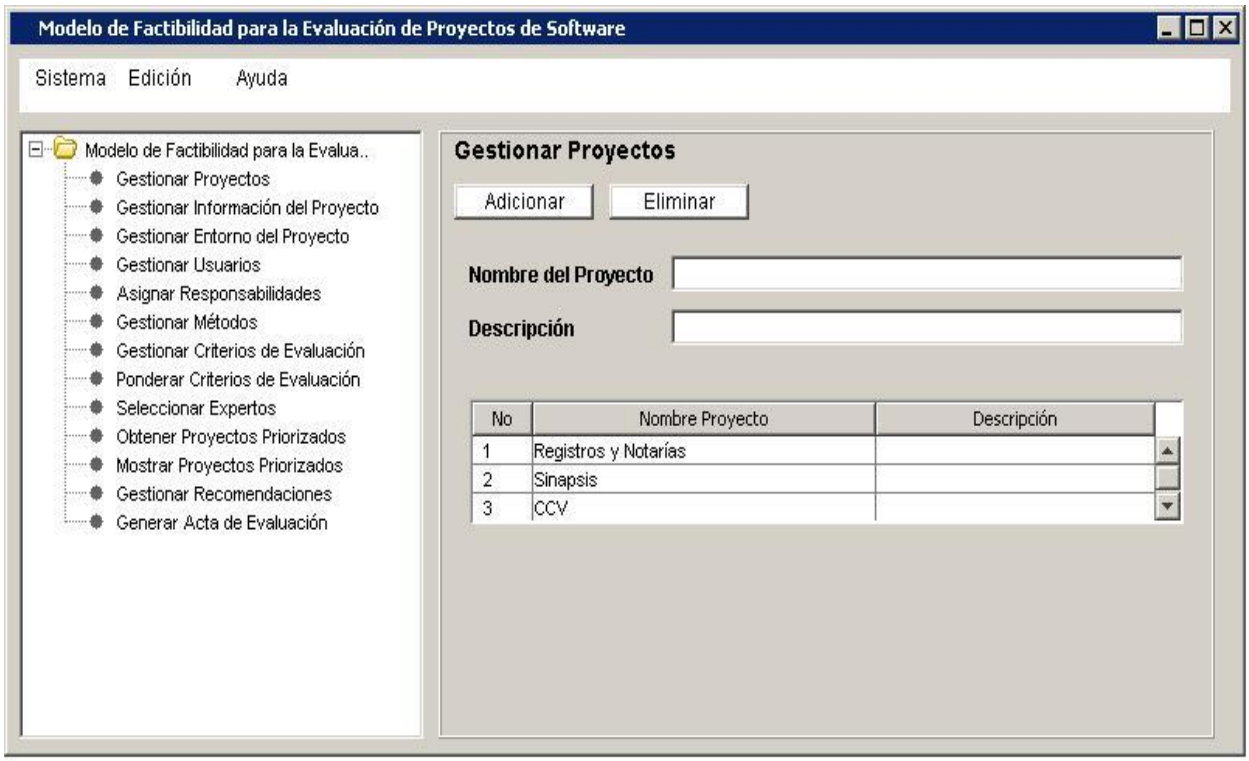
Caso de Uso:	Gestionar Proyectos.
Actores:	Gerente de proyecto
Resumen:	El caso de uso se inicia cuando el Gerente de proyecto necesita calcular la factibilidad de un proyecto. Consiste en que el gerente selecciona la opción correspondiente a ingresar el proyecto en el sistema y llena los campos requeridos para la creación en caso de que no exista el mismo, este usuario también puede acceder a otras opciones, para la eliminación de uno existente en la aplicación.
Precondiciones:	El Gerente de proyecto tiene que estar autenticado.
Referencias	RF.2, RF.3
Prioridad	Crítico

Complejidad	Media
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 Gerente de proyecto selecciona la opción "Gestionar un Proyecto".	2. El sistema muestra la interfaz donde aparecen los campos que debe elegir el usuario: • Adicionar Proyecto. • Eliminar Proyecto.
3. El actor Gerente de proyecto puede realizar una de las siguientes opciones: • Adicionar ver sección **"Adicionar Proyecto"**) • Eliminar (ver sección **"Eliminar Proyecto"**)	
Flujo Normal de Eventos	
Sección "Adicionar Proyecto"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra los campos a llenar para que se introduzcan los datos del nuevo proyecto: • Nombre del Proyecto. • Descripción.
2. El actor Gerente de proyecto introduce los datos necesarios y selecciona la opción "Adicionar".	3. El sistema valida que los datos introducidos son correctos y/o que no hay campos obligatorios vacíos.
	4. El sistema registra el nuevo proyecto y actualiza la interfaz.
Flujos Alternos	
Flujo alterno al paso 3 "Verificar Datos"	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un mensaje de error indicándole al usuario que los datos entrados son incorrectos.
Sección "Eliminar asignación"	
Acción del Actor	**Respuesta del Sistema**

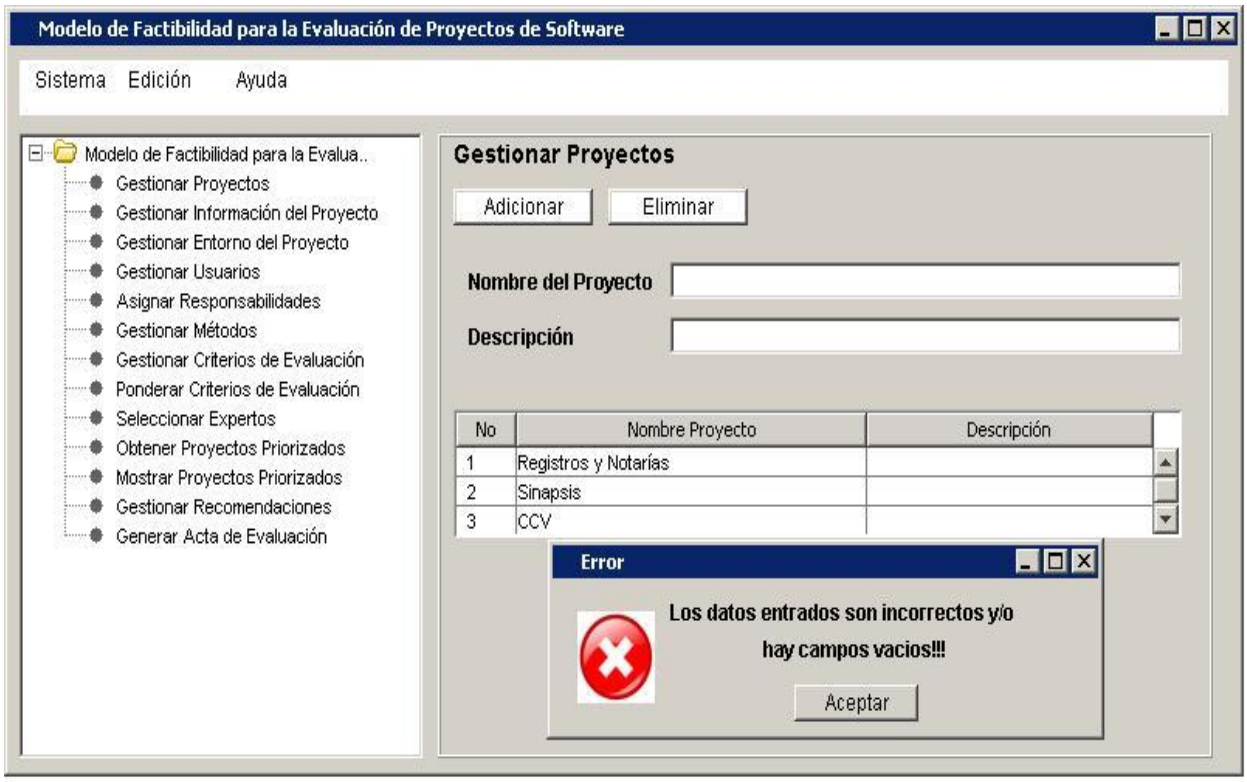
1. El actor selecciona un proyecto.	2. El sistema muestra una interfaz donde se muestran un cuadro de diálogo preguntando si está seguro que desea eliminar el elemento seleccionado.
3. El actor Gerente de proyecto selecciona la opción “Aceptar” del cuadro de diálogo mostrado.	4. El sistema elimina el proyecto seleccionado por el usuario.
	5. El sistema actualiza la eliminación del proyecto, enviando al usuario a la interfaz principal.
Pos-condiciones	• El sistema queda con un nuevo proyecto registrado. • El sistema queda con un proyecto eliminado.

Prototipos de interfaz no funcionales correspondientes a la descripción de casos de uso anterior:

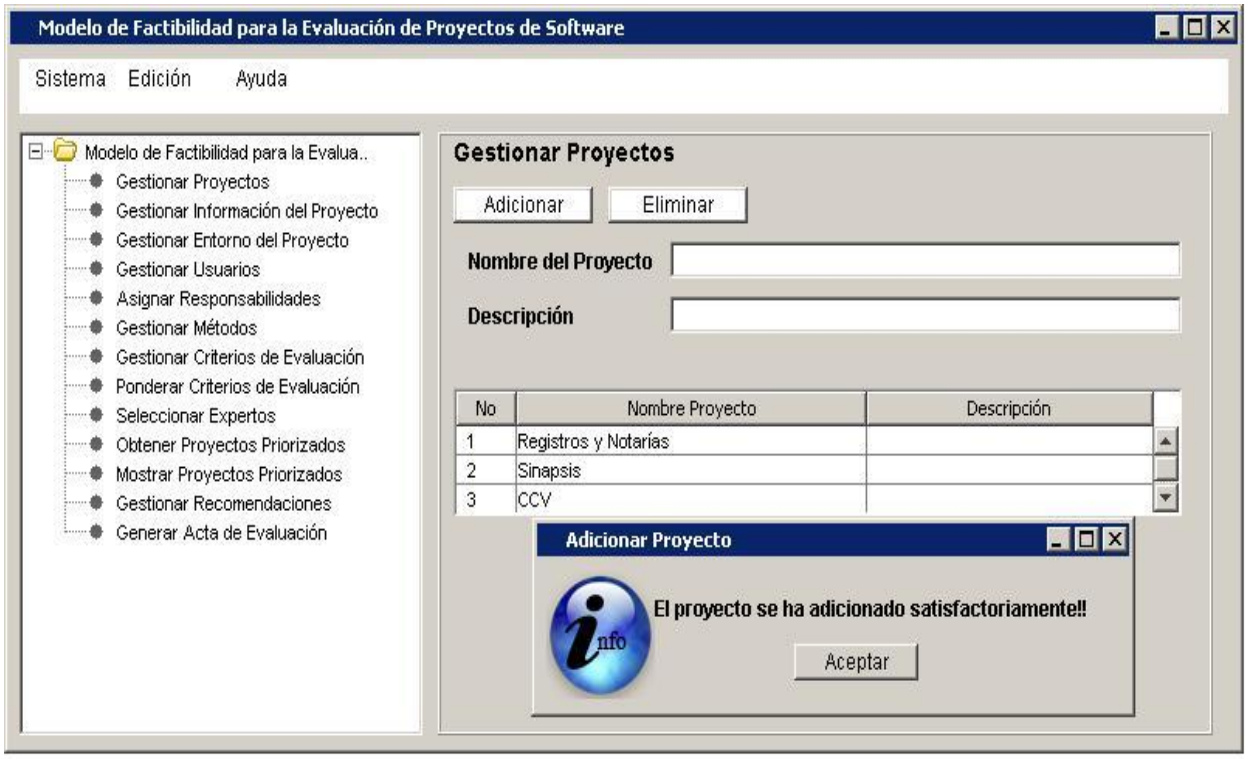
Prototipo Base del Caso de Uso.



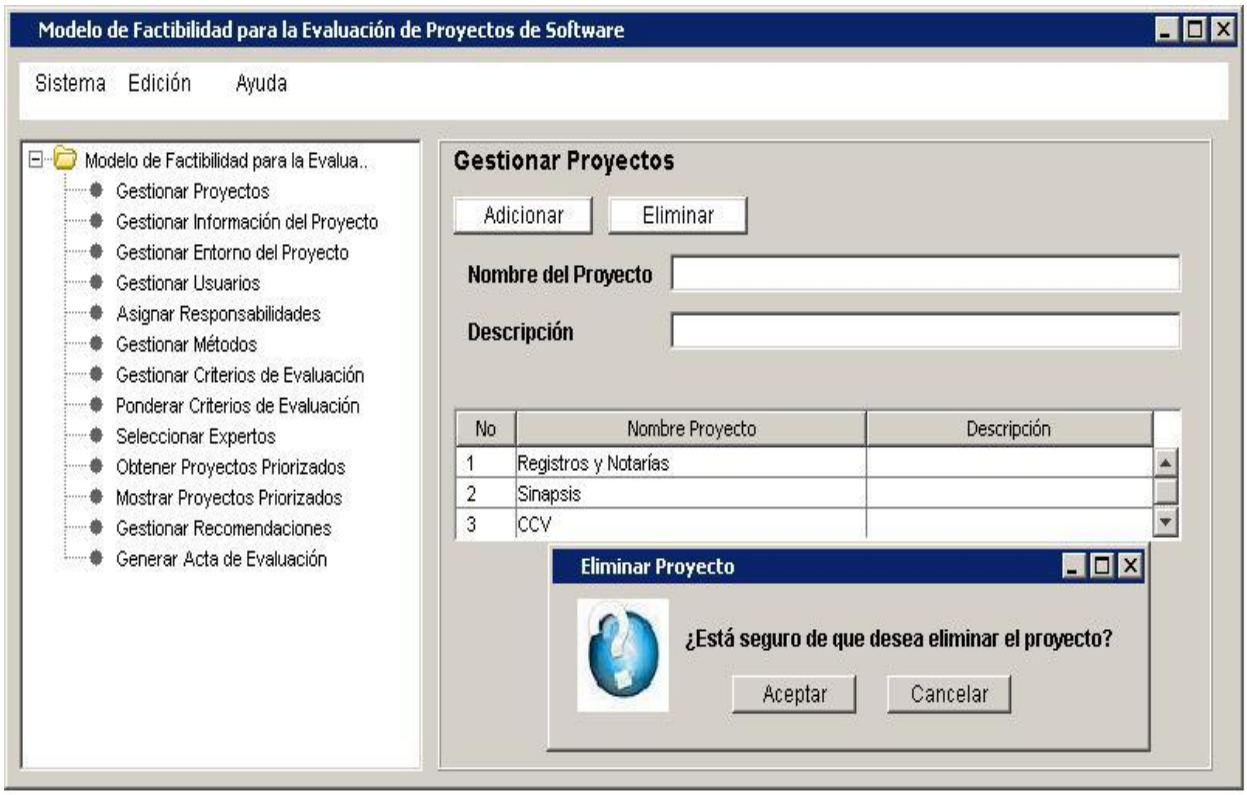
Flujo Alterno.



Sección “Adicionar Proyecto”.



Sección “Eliminar Proyecto”.



2.6 DISEÑO DEL SISTEMA

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

2.6.1 Diagrama de Clases del Diseño

Los diagramas de clases del diseño describen la realización de los casos de uso y al mismo tiempo constituyen una abstracción del modelo de implementación y el código fuente, es una entrada esencial a las actividades de implementación. Con ellos se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Para la realización de dichos artefactos se agruparon los casos de uso del sistema en paquetes por funcionalidades, además se emplearon patrones de diseño tales como, Facade permitiendo la simplificación de los accesos a un conjunto de objetos relacionados y proporcionando un

objeto de comunicación, Builder permitiendo a un objeto construir un objeto complejo, especificando sólo su tipo y contenido, Experto para la asignación de responsabilidades, DAO (Data Access Object) para crear una interfaz común que garantice el acceso a los datos y el MVC (Modelo-Vista-Controlador) permitiendo estructural todo el proceso en 3 capas, Modelo para el encapsulamiento de los datos y las funcionalidades, Vista permitiendo mostrar la información al usuario y Controlador quien permite recibir las entradas y luego traducirlas a solicitudes de servicio para el modelo o la vista.

Los diagramas de clases del diseño pueden ser consultados en el "Documento Modelo de Diseño".

A continuación se muestra el diagrama de clases del diseño Gestión de Proyectos:

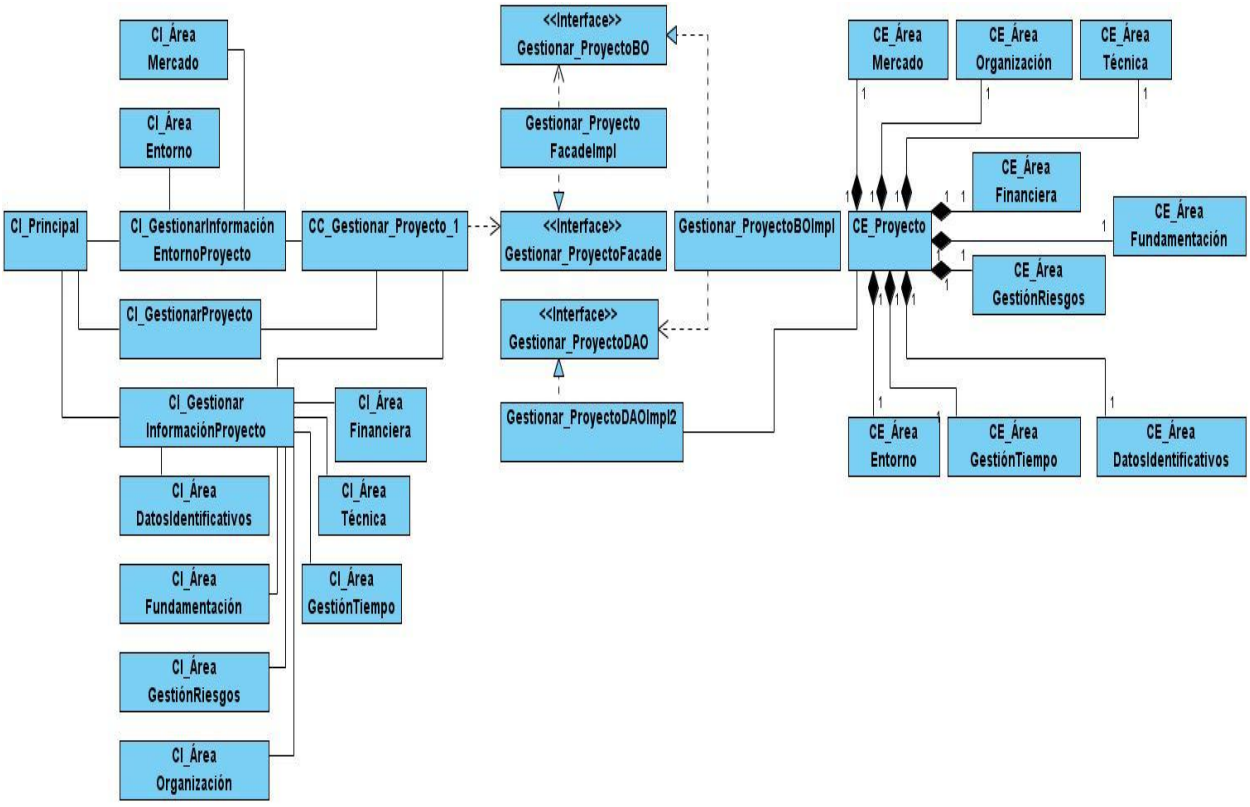


Figura 11 Diagrama de Clases del Diseño Gestión de Proyectos

2.6.2 Diagramas de Secuencia

Los diagramas de secuencia muestran las interacciones de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. Contienen detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

Los diagramas de secuencia del sistema pueden ser consultados en el "Documento Modelo de Diseño". Un ejemplo de los diagramas de secuencia se muestra a continuación:

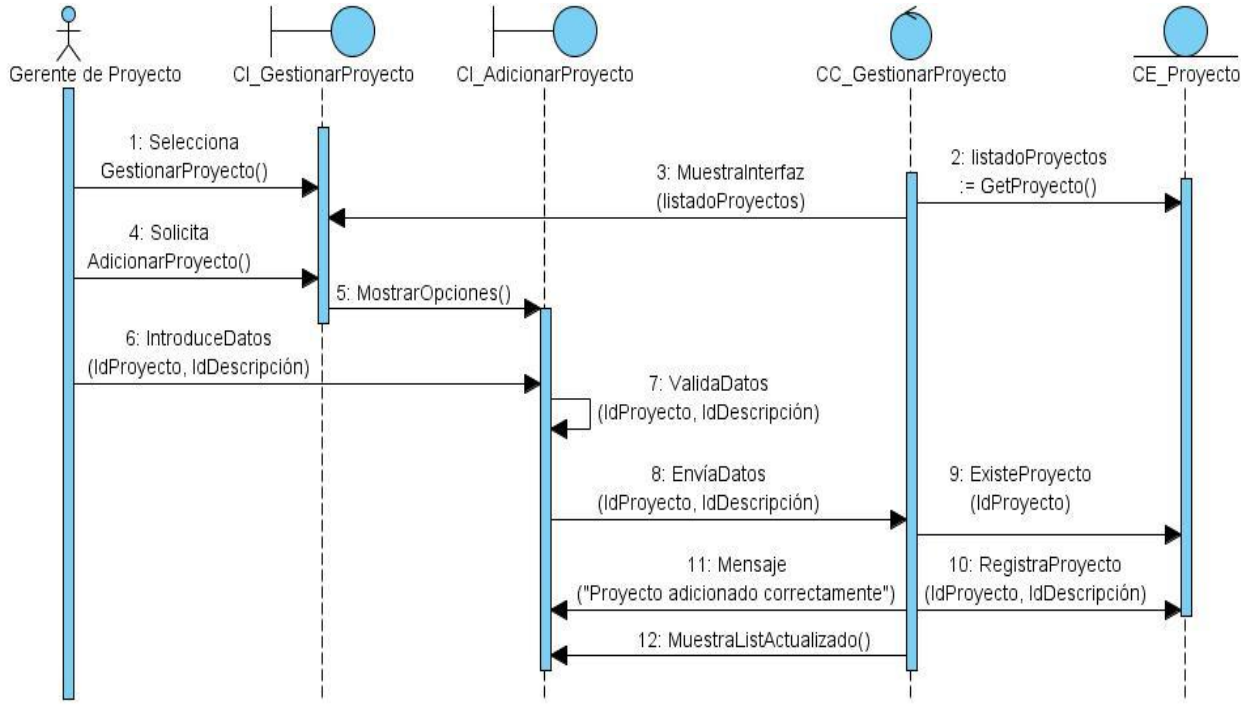


Figura 12 Diagrama de Secuencia CU Gestionar Proyectos (Sección Adicionar Proyectos)

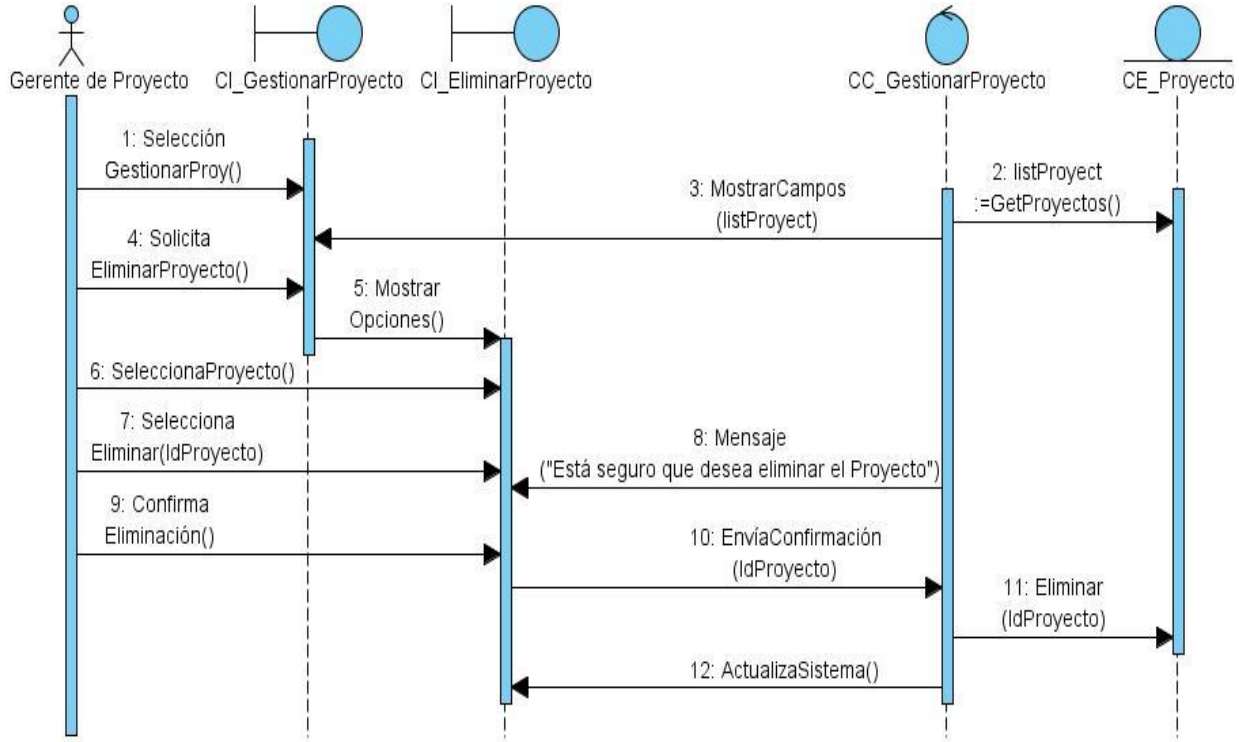


Figura 13 Diagrama de Secuencia CU Gestionar Proyectos (Sección Eliminar Proyectos)

2.7 CONCLUSIONES PARCIALES

En este capítulo se llegó a las siguientes conclusiones:

- La utilización de técnicas para la identificación de requisitos, permitió obtener resultados satisfactorios en la identificación de las funcionalidades con las que deberá contar el sistema.
- Se logró un refinamiento de los requisitos 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.
- La aplicación de patrones, tanto de casos de uso como de diseño, posibilitó obtener artefactos aceptables y evitó que se cometieran errores tradicionales.
- La construcción del modelo del sistema y de diseño permitió obtener los elementos que deberán ser implementados.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN

3.1 INTRODUCCIÓN

En el presente capítulo se hace un análisis de los resultados obtenidos durante el desarrollo de la solución propuesta. Para la validación de los artefactos del modelo del sistema se muestran las listas de chequeo que se le fueron aplicadas, además del acta legal 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 del tamaño de clases.

3.2 VALIDACIÓN DE LOS REQUISITOS

La identificación de los requisitos y la especificación de los de los casos de uso constituyen los principales artefactos obtenidos durante el desarrollo de la ingeniería de requisitos del Modelo de Factibilidad para la Evaluación de Proyectos de Software en la UCI. Con el objetivo de garantizar su factibilidad fueron sometidos a varias iteraciones de revisiones por parte del proyecto calidad de la facultad. Se aplicaron listas de chequeo para los requisitos y para la descripción de los casos de uso (ver “Documento Listas_de_chequeo”). Las no conformidades encontradas durante las iteraciones de la revisión fueron solucionadas por el equipo de análisis (ver “Documento No Conformidades”). La carta de liberación emitida posterior a la revisión de la documentación se encuentra en los anexos (**ver Anexo1**). Además se hicieron varias revisiones por parte de los usuarios, emitiendo posteriormente una Carta de Aceptación como constancia de las revisiones (**ver Anexo2**).

3.3 MÉTRICAS PARA LA CALIDAD DE LA ESPECIFICACIÓN DE LOS REQUISITOS DE SOFTWARE

Los requisitos, una vez definidos, necesitan ser validados. La validación de requisitos tiene como misión demostrar que la definición de estos define realmente las funcionalidades que el usuario necesita o el cliente desea. Es necesario asegurar que el análisis realizado y los resultados obtenidos de la etapa de definición de requisitos son correctos.

Con el objetivo de medir la satisfacción del cliente se aplicaron un conjunto de métricas para la calidad de la Especificación de los Requisitos de Software. A continuación se muestra como se realizó este proceso.

Inicialmente fue necesario calcular el número total de requisitos, para así poder aplicar las métricas que hacen uso de este. (30)

$\mathbf{R_f}$: Número de requisitos funcionales.

$\mathbf{R_{nf}}$: Número de requisitos no funcionales.

$\mathbf{R_t}$: Total de requisitos.

$$\mathrm{R_t} = \mathrm{R_f} + \mathrm{R_{nf}}$$
$$\mathrm{R_t} = \mathbf{34} + \mathbf{10}$$
$$\mathrm{R_t} = \mathbf{44}$$

- **Especificidad**

Para determinar la especificidad (ausencia de ambigüedad) de los requisitos se empleó la métrica basada en la consistencia de la interpretación de los revisores para cada requisito. Cuanto más cerca de 1 esté el valor de $\mathbf{Q_1}$ (grado de especificidad de los requisitos), mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad de la especificación de los requisitos. (30)

$\mathbf{R_{ii}}$: Número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.

$$\mathrm{Q_1} = \frac{\mathrm{R_{ii}}}{\mathrm{R_t}}$$
$$\mathrm{Q_1} = \frac{\mathbf{41}}{\mathbf{44}}$$
$$\mathrm{Q_1} = \mathbf{0.93}$$

- **Corrección**

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1 e indica un alto nivel de corrección en la definición de los requisitos. Este valor se calcula como se muestra a continuación: (30)

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

$\mathbf{R_c}$: Número de requisitos que se han validado como correctos.

$\mathbf{R_{nv}}$: Número de requisitos que no se han validado como correctos todavía.

$$\mathrm{Q_2} = \frac{\mathrm{R_c}}{\mathrm{R_c} + \mathrm{R_{nv}}}$$
$$\mathrm{Q_2} = \frac{\mathbf{44}}{\mathbf{44} + \mathbf{0}}$$
$$\mathrm{Q_2} = \mathbf{1}$$

- **Compleción**

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1 e indica un alto nivel de completitud en la definición de los requisitos. Este valor se calcula como se muestra a continuación: (30)

$\mathbf{n_A}$ **:** Número de requisitos completos.

$\mathbf{n_B}$ **:** Número de requisitos pobremente especificados**.**

$$Q_3 = \frac{n_A}{n_A + n_B}$$

$$Q_3 = \frac{44}{44 + 0}$$

$$Q_3 = 1$$

- **Comprensión**

La comprensión de los requisitos se determinó a partir de la relación que se muestra a continuación. El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1. (30)

$\mathbf{R_{bc}}$ **:** Número de requisitos que todos los revisores entienden.

$$Q_4 = \frac{R_{bc}}{R_t}$$

$$Q_4 = \frac{44}{44}$$

$$Q_4 = 1$$

- **Consistencia interna**

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1 y expresa que no existen subconjuntos de requisitos contradictorios. El valor óptimo de esta métrica es el más cercano a 1. (30)

$\mathbf{n_u}$: Número de requisitos especificados.

$\mathbf{n_n}$: Número de requisitos en conflicto con otros requisitos en la especificación.

$$Q_5 = \frac{n_u - n_n}{n_u}$$

$$Q_5 = \frac{44 - 0}{44}$$

$$Q_5 = 1$$

- **Consistencia externa**

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1 y expresa que ninguno de los requisitos está en contradicción con lo expresado en documentos de nivel superior, o sea los requisitos del software no pueden contradecir los requisitos del sistema. El valor óptimo de esta métrica es el más cercano a 1.

n_{ec} : Número de requisitos que son consistentes con otros documentos. (30)

$$Q_6 = \frac{n_{ec}}{R_t}$$

$$Q_6 = \frac{44}{44}$$

$$Q_6 = 1$$

- **Estabilidad**

Para medir la estabilidad de los requisitos de software, en el presente trabajo se aplicó la métrica propia para esto, la cual ofrece valores entre 0 y 1. El mejor valor de $\mathbf{E}$ es el más cercano a 1. La estabilidad de los requisitos se calcula de la siguiente forma:

$\mathbf{E}$: Valor de la estabilidad de los requisitos. (30)

$\mathbf{R_t}$: Total de requisitos.

$\mathbf{R_m}$: Número de requisitos modificados, que se obtienen como la sumatoria de los requisitos insertados, modificados y eliminados.

$$E = \left[\frac{R_t - R_m}{R_t}\right]$$

$$E = \frac{44 - 4}{44}$$

$$E = 0.90$$

En la siguiente gráfica se pueden visualizar los resultados obtenidos al aplicar estas métricas:

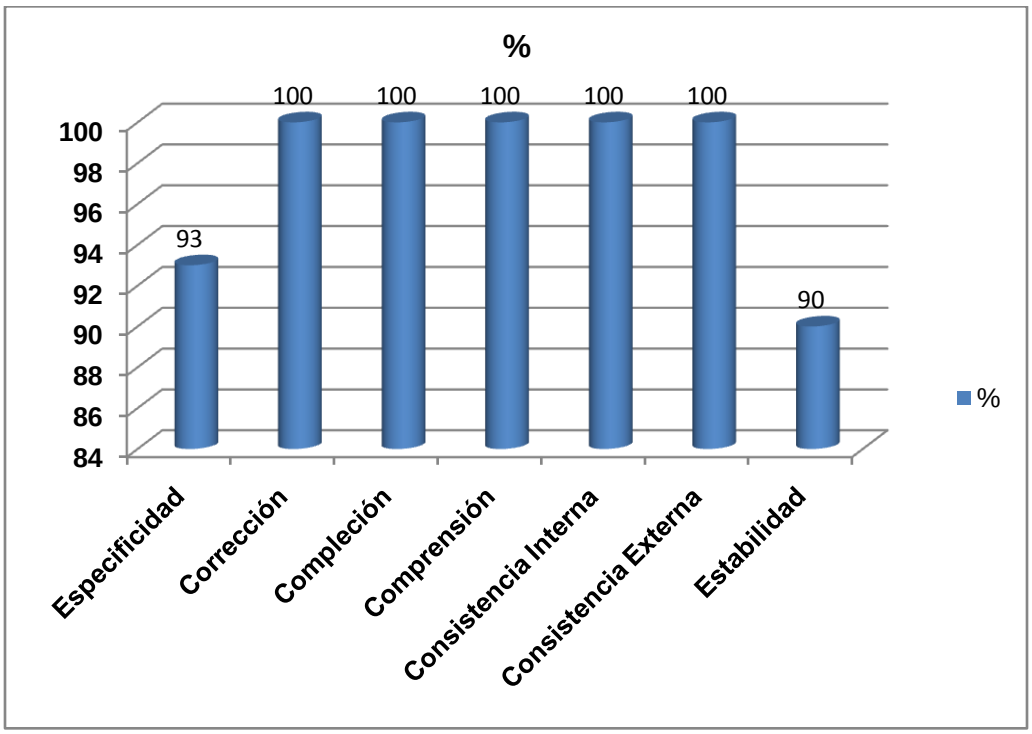


Figura 14 Resultados de las métricas de la calidad de la Especificación de Requisitos de Software

Para las métricas especificidad, corrección, comprensión, consistencia interna y externa se recomienda que los resultados estén más cercanos a 1 para ser clasificados como óptimos. El

umbral para la métrica de compleción se recomienda que sea un peso aproximado a 0.7, aunque si este es 1 la métrica es clasificada como completa. En el caso de la estabilidad se clasifica en alta (0.90<=**E**<=1), en media (0.80<=**E**<0.90) y en baja (0.7<=**E**<0.80). Con los resultados obtenidos se demostró que el proceso de especificación de los requisitos cuenta con una alta calidad y además ello demuestra que se hizo un buen trabajo durante el proceso de captura de requisitos y que hay entendimientos comunes entre clientes y equipo de desarrollo.

3.4 MÉTRICAS PARA LA EVALUACIÓN DE LA CALIDAD EN LOS DIAGRAMAS DE CASOS DE USOS

Las métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema definen cuatro atributos: (30)

- Completitud: permite determinar el grado en que se ha 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 rehúso del artefacto.

Estos atributos presentan un significado determinado de acuerdo con el tipo de artefacto y al 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: (30)

No.	Atributo.	Métricas.
1.	Completitud	Número de casos de uso que no tiene 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	14	Todos poseen una descripción resumida.	0%
2	14	Ninguno presenta requerimientos omitidos, al menos están relacionados con 1 requerimiento.	0%
3	14	No se detectó ningún caso de uso que no presentara su descripción extendida.	0%
4	14	Todos están redactados en función del responsable que le corresponde a cada uno.	0%
5	14	No se detectó ningún caso de uso que no describiera condiciones de excepción relevantes.	0%
6	14	Todos han sido clasificados según los diferentes tipos de prioridad.	0%
7	14	Ninguno presentó problemas en cuanto al nombre dado, ya que todos los nombres representan una expresión verbal en infinitivo describiendo alguna funcionalidad relevante y significativa.	0%
8	14	Todos representan correctamente la interacción con un actor.	0%
9	14	Ninguno presentó acciones del flujo de eventos que estuvieran asignados a un responsable que no le corresponde.	0%

10	14	Todos fueron aceptados.	0%
11	14	Todos poseen sus descripciones del flujo básico y de flujos alternos separadas.	0%
12	14	Todos poseen un usuario responsable.	0%
13	14	Todos los casos de uso en que los requerimientos fueron representados son comprensibles por el usuario.	0%
14	14	Se detectó un caso de uso que debe ser modificado para adecuarlo a la funcionalidad del sistema.	7.14%
15	14	Ninguno tuvo que ser modificados para mejorar el proceso actual.	0%
16	14	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:

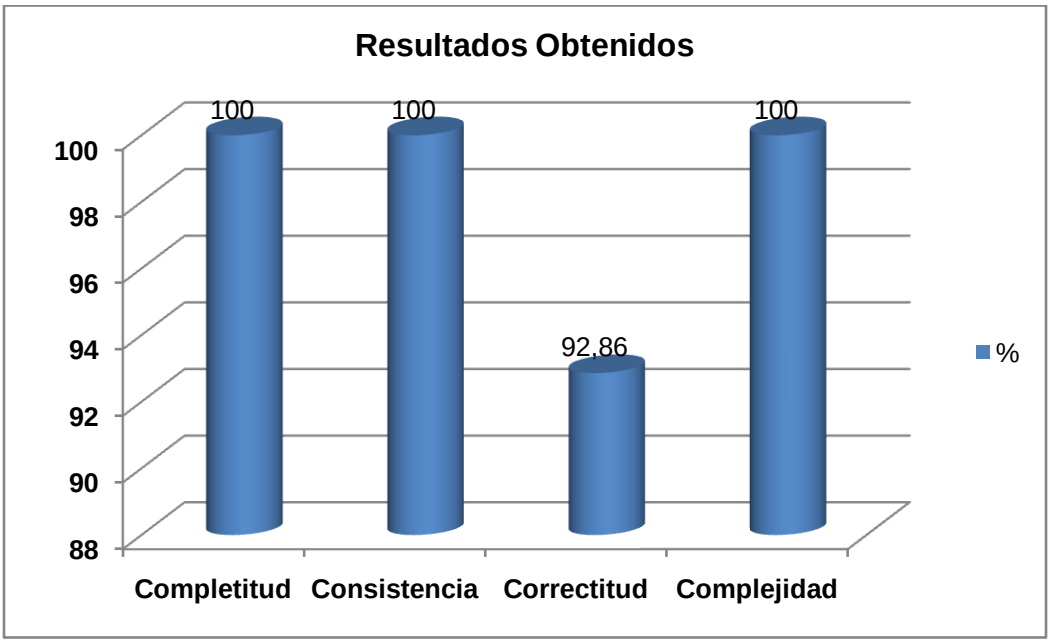


Figura 15 Resultados de la evaluación de la calidad en los diagramas de casos de usos

Como resultado de aplicar esta métrica se pudo observar la calidad y funcionalidad del diagrama de casos de uso, obteniéndose un 100% tanto en completitud, consistencia y complejidad y no siendo así en Correctitud donde se obtuvo un 92.86% debido a que existe un caso de uso que deberá ser redefinido para que cumpla con necesidades específicas del cliente.

3.5 MÉTRICAS DE DISEÑO

3.4.1 Métricas de Cohesión

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
Gestionar_ProyectoBOImpl	2
Gestionar_CriterioBOImpl	3
Gestionar_UsuarioBOImpl	4
Gestionar_Proyecto_2BOImpl	2
GestionarMétodosEvaluaciónBOImpl	3
Gestionar_ProyectoDAOImpl2	3
CC_Gestionar_Usuario	1
CC_GestionarProyecto_2	1
CC_GestionarMétodosEvaluación	1

Según los datos de las clases analizadas se tiene que:

número de elementos = 9

número de súper adhesivos(i) = 0

número de adhesivos(i) = 6

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

$$CFD = \frac{6}{9} = 0.67$$

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 sistema posee una cohesión funcional alta para un 67% de fortaleza.

3.4.2 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.

Clases	No. Atributos	No. Operaciones
Principal	14	14
ÁreaEntorno	16	1
ÁreaMercado	6	1
GestionarProyecto	2	3
GestionarInformaciónProyecto	2	6
ÁreaDatosIdentificativos	3	1
ÁreaFundamentación	4	1
ÁreaGestiónRiesgos	6	1
ÁreaGestiónTiempo	2	1
ÁreaOrganización	1	1
ÁreaFinanciera	1	1
ÁreaTécnica	6	1
Gestionar_Proyecto_1	1	12
Gestionar_ProyectoFacadeImpl	1	12
Gestionar_ProyectoBOImpl	1	12
Gestionar_ProyectoDAOImpl2	1	12
Proyecto	9	18
CE_ÁreaMercado	6	12
CE_ÁreaOrganización	1	2
CE_ÁreaTécnica	6	12
CE_ÁreaFinanciera	1	2
CE_ÁreaFundamentación	4	8
CE_ÁreaDatosIdentificativos	3	6
CE_ÁreaGestiónRiesgos	6	12
CE_ÁreaGestiónTiempo	2	4
CE_ÁreaEntorno	16	32
CI_PonderarCriterios	2	2
CI_GestionarCriterios	3	6

CC_Gestionar_Criterio	1	8
Gestionar_CriterioFacadeImpl	1	8
Gestionar_CriterioBOImpl	1	8
Gestionar_CriterioDAOImpl	1	8
CE_Criterio	5	10
CI_Autenticar	2	1
CI_GestionarUser	5	4
CI_AsignarResponsabilidades	1	3
CI_SeleccionarExperto	1	2
CC_Gestionar_Usuario	1	8
Gestionar_UsuarioFacadeImpl	1	8
Gestionar_UsuarioBOImpl	1	8
Gestionar_UsuarioDAOImpl	1	8
CE_Usuario	3	6
CE_Rol	5	10
CI_MostrarProyectosPriorizados	1	1
CI_GenerarActaEvaluaciónProyectos	1	2
CI_GestionarRecomendaciones	2	4
CI_ObtenerProyectosPriorizados	3	5
CI_FlujoDeCaja	6	2
CI_BCR	2	2
CI_RVAN	1	2
CI_ValorActualNeto	4	2
CI_PRI	2	2
CI_Vista	4	0
CC_GestionarProyecto_2	1	10
Gestionar_Proyecto_2Impl	1	10
Gestionar_Proyecto_2BOImpl	1	10
Gestionar_Proyecto_2DAOImpl	1	10
CE_Metaevaluador	3	6
CI_GestionarMétodosEvaluación	3	5
CC_GestionarMétodosEvaluación	1	5
GestionarMétodosEvaluaciónFacadeImpl	1	5
GestionarMétodosEvaluaciónBOImpl	1	5
GestionarMétodosEvaluaciónDAOImpl	1	5
CE_Evaluaciones	1	2
CE_MétodosEvaluación	1	1
CE_FlujoCaja	5	10
CE_ValorActualNeto	4	8
CE_RVAN	1	2
CE_BCR	2	4
CE_PRI	2	4

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.

En las 70 clases presentadas se obtuvo un promedio de 3 atributos y 6 operaciones o métodos por clase. Atendiendo a los resultados arrojados por la métrica se tiene que:

Umbral	Tamaño	Cantidad de Clases
<=20	Pequeño	69
>20 y <=30	Mediano	0
>30	Grande	1

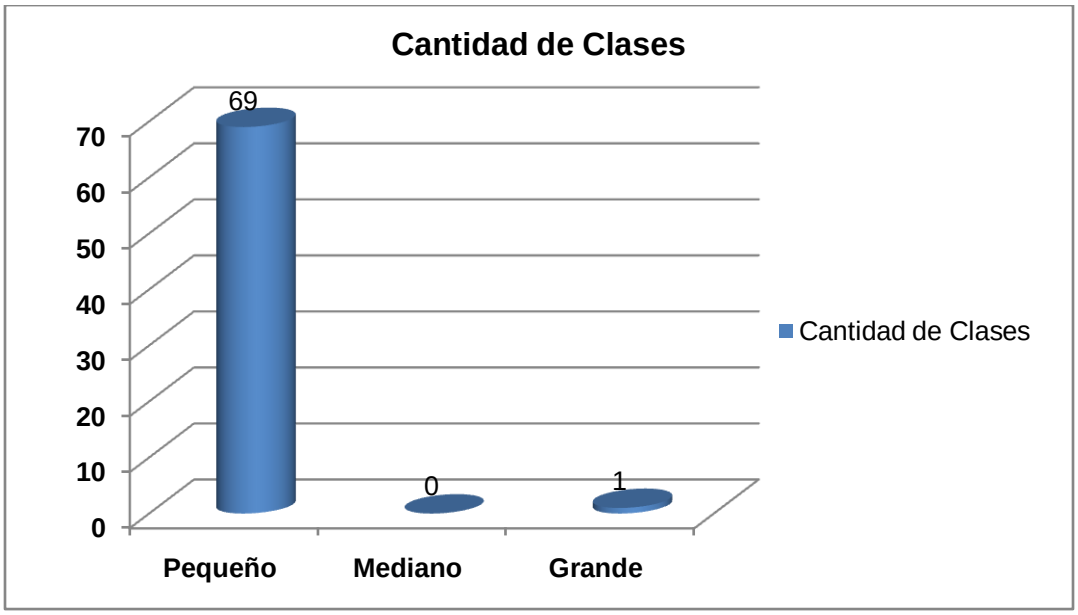


Figura 16 Número de clases por categorías

Con esta métrica se pudo comprobar que atendiendo a la cantidad de operaciones, el 98.57 % de las clases diseñadas están consideradas como pequeñas, lo que facilitará el proceso de construcción del sistema.

3.6 CONCLUSIONES PARCIALES

En este capítulo se analizaron los resultados obtenidos durante el desarrollo del trabajo, es decir, se avaluaron todos los artefactos obtenidos, arribándose a las siguientes conclusiones:

- Las actas de aceptación del cliente y liberación por parte del equipo de calidad de la facultad demostraron que tanto los requisitos 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 Documento de Especificación de Requisitos demostró que estas se corroboraron con las necesidades de clientes/usuarios finales y que son correctas las interpretaciones por parte del equipo de desarrollo, así como la aplicación de las métricas para la calidad de la funcionalidad del diagrama de casos de uso demostró que se construyó un diagrama con calidad.
- Las métricas aplicadas a los elementos del diseño permitió comprobar la calidad de los mismos y además de que posibilitó tener una visión de la complejidad que tendrá el proceso de desarrollo del sistema.

CONCLUSIONES GENERALES

Se arribó a las siguientes conclusiones:

- Se realizó un estudio de las tendencias en las disciplinas a desarrollar, las herramientas a utilizar para modelar, la metodología a tener en cuenta, los patrones de casos de uso y de diseño, así como las métricas para evaluar la calidad de la solución propuesta. Esto permitió la preparación teórica que sustenta el desarrollo del trabajo.
- El estudio de los principales conceptos y aspectos que propone el Modelo de Factibilidad para la Evaluación de Proyectos de Software, junto a la interacción directa con los clientes del sistema y la aplicación de algunas técnicas para capturar los requisitos, permitió que se obtuvieran resultados satisfactorios en la identificación de las funcionalidades que debe cumplir el sistema.
- La elaboración de los artefactos del modelo 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 con los requisitos especificados.
- Se evaluó la calidad de los principales artefactos generados, los requisitos, los casos de uso del sistema y el diseño, mediante métricas definidas por autores reconocidos y la revisión por parte del proyecto de calidad de la facultad, lo que permitió confirmar la realización de artefactos confiables y con calidad, que garantizarán un entendimiento de las funcionalidades a los desarrolladores permitiendo la futura implementación.

RECOMENDACIONES

Se recomienda:

- Realizar un seguimiento de los requisitos de software durante las posteriores fases de desarrollo, aplicando la Administración de estos que propone la Ingeniería de Requisitos, para garantizar una gestión y control de los cambios.
- Continuar con la elaboración de los restantes artefactos que genera el flujo de análisis y diseño, que facilitan continuar con el desarrollo del sistema.
- Tener en cuenta la posibilidad de incluir nuevos métodos de evaluación de proyectos al metaevaluador, permitiendo obtener una estructura evaluadora actualizada con las nuevas tendencias.
- Realizar la implementación de la propuesta que se presenta en este trabajo, con el fin de obtener al menos una versión del producto.

BIBLIOGRAFÍA

1. **Palacio, J.** Origen de la Gestión de Proyectos. [En línea] 2006. http://www.navegapolis.net.

2. **Pereña Brand, J.** *Dirección y Gestión de Proyectos.* 1996.

3. **Peña Abreu, Marieta y Bertamí Barrios, Beatriz.** *Propuesta de Modelo de Perfactibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas.* Ciudad de la Habana, UCI Facultad3 : s.n., 2009.

4. **Gestiopolis.** Gestiopolis. [En línea] http://www.gestiopolis.com/recursos/experto/catsexp/pagans/ger/no12/factibilidad.htm.

5. **Urda, B.M.O.** *Gerencia de Proyectos de Ciencia e Innovación Tecnológica.* Ciudad de la Habana, Cuba : Folleto curso de postgrado. CITMA, 1998.

6. **Pressman, Rogen S.** *Ingeniería de Software." Un Enfoque Práctico.".* 2005. 5ta Edición.

7. **IEEE.** *Standards Collection: Sofware Engineering.* 1993.

8. **Boehm, Barry W.** *Software Engineering.* 1976.

9. —. *Software Engineering Economics.* NY, USA : s.n., 1981.

10. **Robertson, Suzanne y Robertson, James.** *Mastering the Requirements Process.* 2006. Second Edition.

11. **Thayer, R y Dorfman, M.** *Standards, Guidelines and Examples on System and Software Requirements Engineering.* 1990.

12. **Kotonya, G y Sommerville, I.** *Requirements Engineering: Processes and Techniques.* 2000.

13. **Durán Toro, Amador y Bernárdez Jiménez, Beatriz.** *Metodología para la Elicitación de Requisitos de Sistemas Software.* 2000.

14. **Menéndez, R y Asensio, B.** Metodologías de desarrollo de software. [En línea] 2005. http://www.um.es/docencia/barzana.

15. **Gómez, J y Gómez, P.** ¿Metodologías?… si pero, ¿cuál? [En línea] 2006. http://www.versioncero.com/articulo/469/metodologiasi-pero-cual.

16. **Jacobson, Ivar, Booch, G y Rumbaugh, J.** *El Proceso Unificado de Desarrollo de Software.* Madrid, España : s.n., 2000.

17. **Torres, Francisco.** [En línea] 2006. http://giga4.es/archivos/raiz/moviendoarchivos/2008/febrero/presentacion_tac.pdf.

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

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

20. **Larman, Graig.** *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 2004.

21. **Durocher, E.** *Business process Managements .* 2007.

22. **Ramos González, Juan José.** *PML-A modeling Language for Physical Knowledge Representation.* Barcelona : s.n.

23. **Kendall, K.** *Análisis y Diseño de Sistemas.* 1997.

24. **Boost Productivity with Innovative and Intuitive Technologies.** Boost Productivity with Innovative and Intuitive Technologies. [En línea] 2007. http://www.visual-paradigm.com/aboutus/10reasons.jsp.

25. **RATIONAL.** www.rational.com. [En línea] http://www.rational.com.

26. **SPARX System Pty L.** SPARX System. [En línea] 2000-2007. http://sparxsystems.com.ar/products/ea.html..

27. **Wilson, Leslie B.** *Comparative Programming Languages.* 1993. 2da Edición.

28. **Geetanjali, A y Balasubramaniam, A.** *Programación Microsoft C#.* Cidad de La Habana : Félix Varela, 2007.

29. **Overgaard, Gunnar y Palmkvist, Karin.** *Use Cases Patterns and Blueprints.* 2004.

30. **Overmyer, Davis S.** *Identifying and measuring quality in software requirements specification.* Los Alamitos, California. : s.n., 1993.

ANEXOS

ANEXO 1. 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 Modelo de Factibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas y haberse detectado un promedio de 8 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

Decanato Facultad

ANEXO 2. CARTA DE ACEPTACIÓN DEL CLIENTE

Universidad de las Ciencias Informáticas

Ciudad de la Habana, Viernes, 14 de mayo de 2010.

Año 52 de la revolución

Carta de Aceptación del Cliente.

A quién pueda interesar:

Luego de efectuar un grupo de revisiones a los artefactos generados en la tesis análisis y diseño del Modelo de Factibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas, se verificó que los mismos cumplían las necesidades y cubrían las funcionalidades que el sistema debe permitir, además se verificó que no hay requerimientos omitidos, los nombres dados representan una expresión verbal que reflejan alguna funcionalidad relevante y significativa, las interfaces concebidas son amigables lo cual permite un mejor entendimiento y desenvolvimiento del usuario.

Para que conste la Aceptación de la solución propuesta firma la presente, como usuario que ha hecho uso del Modelo de Factibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas el cual responde a las necesidades por las cual fue concebido mejorar la factibilidad técnica y económica

Este trabajo es parte de un trabajo de investigación que tributa a una tesis de Maestría en Gestión de Proyectos Informáticos, relacionada con el análisis de factibilidad de proyectos de desarrollo de software.

Dr.C. Pedro Yobanis Piñero Pérez

Director de la Dirección Técnica de la Producción

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

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

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.

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.

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

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

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