Universidad de las Ciencias Informáticas

Facultad 6

Título: Plugin de “Visual Paradigm for UML” para la aplicación del Método de Estimación UCI”.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Autores: Mayelin Timoteo Guerra

Ayeris Lizandra Navarro González

Tutores: Ing. Adaily Hernández Carballé

Ing. Armando Robert Lobo

La Habana, Junio 2012

“Año 54 de la Revolución”

"Si quieres hacer algo grande en tu vida, debes recordar que la timidez está solo en la mente. Si piensas con timidez, tus actos serán similares. Pero si piensas con seguridad, actuaras de la misma forma. Por ello nunca dejes que la timidez conquiste tu mente".

Bill Gates

Declaración de Autoría

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

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

_____	_____
Mayelin Timoteo Guerra.	Ayeris Lizandra Navarro González.
Autores	**Autores**
_____	_____
Ing. Armando Robert Lobo.	Ing. Adaily Hernández Carballé.
Tutor	**Tutora**

AUTORES:

Mayelin Timoteo Guerra

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: mtimoteo@estudiantes.uci.cu.

Ayeris Lizandra Navarro González

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail:alnavarro@estudiantes.uci.cu.

TUTORES:

Ing. Armando Robert Lobo.

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: arobert@uci.cu.

Ing. Adaily Hernández Carballé

Universidad de las Ciencias Informáticas,

Ciudad de la Habana, Cuba

E-mail: acarballe@uci.cu.

Agradecimientos

Quiero agradecer primeramente a mi abuelo, gracias por ese amor infinito que me brindaste desde pequeña y estar siempre a mi lado hasta el último día de tu vida.

A mi madre Zenaida, por ser la mujer más luchadora que conozco, por el amor y la confianza que no me faltaron nunca, por creer y estar pendiente de mí. Por ser mi razón de ser. Ete te quiero mucho.

A mami por ser mi abuelita del alma porque su ternura y preocupación hacia mí son infinitas.

A mis hermanas Magdelín y Maikelín, gracias por el amor que me han dado y por la confianza que ha tenido en mí.

A mi sobrinito Magdiel, aunque no lo conozco sé que voy a ser la mejor tía, porque cada día aspiro a ser mejor solo para servirle de ejemplo.

A Rafael por ser la persona que siempre ha estado a mi lado durante estos años, por apoyarme en todo, por ese amor, esa dedicación hacia mí y por aguantarme mis malcriadeces. Te quiero mucho Tito.

A mi cuñi del alma Yoe y su familia, por su apoyo incondicional, mi más sincera gratitud.

A familia por estar siempre al tanto de lo que sucede en mi vida y por estar pendientes de mi carrera universitaria y darme ánimos para seguir adelante.

A mi prima Haydee por ser una hermana y a quien le debo tanto.

A primos Yuniel, Yunisdel y Juan por todo su amor y por tenerlos siempre conmigo en la infancia.

A mis tutores Adaily y Armando Robert Lobo, por enseñarnos tanto en el desarrollo de esta investigación y brindarnos toda la información necesaria para la realización de la misma, sin ellos no sería posible este logro y por el tiempo dedicado hacia nosotras.

A Reyni por enseñarme que si se puede que lo último que se pierde es la esperanza y que ante todo tenemos que confiar en lo que somos capaces de hacer, gracias por soportarme.

A la profe Yanelis Benítez, sin su ayuda hoy no hubiese cumplido mi sueño.

A todas las amistades que hemos ganado en esta escuela, muchas de las cuales tal vez no volvamos a ver por la distancia que nos separa, pero que siempre las tendremos presentes en nuestros corazones, que de una forma u otra me han ayudado a formar parte de sus vidas y que de cierta forma he compartido muy buenos momentos y he tratado de ser igual para ellos, por tener el valor de rectificarme, por

aconsejarme y apoyarme siempre que lo he necesitado. Especialmente a Eli más que amiga es mi hermanita, por estar siempre cuando la necesite y preocuparse por mí, Alexey por ser un gran amigo y ayudarme mucho, Rauli por ser el hermanito de la universidad, Fi por ser la mejor secretaria y a Mayeleiny por ser otra amiga que me dio la universidad, por estar juntas en los buenos y malos momentos que hemos pasados, gracias por darme ánimos para seguir adelante y por no perder la calma cuando yo si lo hacía. Espero haber sido para ustedes una gran amiga también.

A nuestro Tribunal por sus críticas constructivas que hicieron posible la culminación de este trabajo con mayor calidad.

A toda personas de Juana, gracias por estar pendiente de mi.

A mi compañera y más que eso mi amiga Ayeris ha sido un gran apoyo durante los 5 años de la carrera y por hacer posible este sueño junto a mí.

A la Revolución por permitirnos la educación y formación profesional.

A la UCI por ser nuestro hogar, por darnos la posibilidad de crecernos como personas, por educarnos, por darnos tantos amigos.

A todas estas personas especiales para mí, quiero darle las gracias por brindarme toda su ayuda, apoyo y compresión, alentándome de esta forma a lograr hacer mi sueño realidad.

Maye

En primer lugar quiero agradecerle a mi madre, por ser la mejor madre del mundo por quererme mucho, mucho, mucho ,por estar siempre a mi lado ,por ser mi guía, mi amiga , eres parte de mi, te amo mucho.

A mi tío lindo por ser el mejor hombre del mundo por ser tan bueno, por darme siempre su amor, por estar en todo momento conmigo, por quererme como una hija, te amo tío.

A mi esposo por compartir momentos buenos y malos conmigo, por confiar en mí, por darme todos mis gustos, te quiero titi.

A familia por estar siempre al tanto de lo que sucede en mi vida y por estar pendientes de mi.

A mis hermanitos del alma Maite, Mara Miguel por estar siempre a mi lado cuando más los necesite.

A mi amiga Mayeleiny por estar siempre dándome buenos consejos.

A mi amiga Yadiri que aunque nunca hemos convivido juntas siempre ha sido una de mis mejores amigas te quiero yadi.

A mi amigo Alexey por ser tan bueno y ayudarme tanto cuando más lo necesite, gracias papa por todo.

A mis tutores, por enseñarnos tanto en el desarrollo de esta investigación y brindarnos toda la información necesaria para la realización de la misma, sin ellos no sería posible este logro y por el tiempo dedicado hacia nosotras.

A mi profe del alma reini por ser el mejor pofesor del mundo te quiero reini.

A Fi la secretaria de mi facultad por ser tan buena.

A mis compañeros de estudio, a los viejos y a los nuevos, a los que ya no están, quienes me ayudaron, apoyaron y aconsejaron en los momentos más difíciles, con los que compartí una magnífica etapa de mi vida.

A compañera de tesis y mi amiga maye por compartir momentos difíciles conmigo ya que ha sido un gran apoyo durante los 5 años de la carrera, por compartir este logro juntas.

Ayeris

Dedicatoria

Dedico el presente trabajo de diploma a quienes le debo la vida a mi mamá, a mi abuela y a mi abuelo por el apoyo, confianza y el amor que siempre me han brindado.

A mis hermanas y mi sobrino por ser el motivo más grande de estar aquí.

A mi esposo por su amor y dedicación hacia mí. Con ellos comparto este triunfo, ya que a ellos se lo debo. Para todos ustedes con cariño y amor.

Maye

Dedico esta tesis a mi mamá, mi tío, mi esposo, mi padre, a mi abuela, por el amor que siempre me han brindado.

Ayeris

Resumen

En el proceso de desarrollo de software es fundamental realizar una buena estimación con el fin de predecir el valor de las variables concretas dentro de un proyecto. Además ver si se cuenta con el tiempo, el esfuerzo o los recursos necesarios para llevarlo a cabo y de esta forma determinar si es viable o no su realización. La presente investigación tiene como objetivo implementar un *plugin* para*"Visual Paradigm for UML"*que permita realizar la estimación de un proyecto aplicando el Método de Estimación UCI. Con este fin se realizó la caracterización del método y fueron definidas las tecnologías y herramientas a utilizar en el desarrollo del *plugin*. Finalmente, se obtuvo el *plugin* que brinda las funcionalidades necesarias para realizar la estimación que permite reducir el tiempo, el costo y el esfuerzo en el proceso de desarrollo de software, así como la especificación de un perfil UML a aplicar en el modelado para obtener la información necesaria de las métricas para realizar la estimación.

Palabras claves:

"Visual Paradigm for UML", Método Estimación UCI, *plugin*, perfil UML, métricas.

Índice de Tablas

Introducción

El proceso de gestión de un proyecto de software inicia con un conjunto de actividades, generalmente se denominan planificación del proyecto, el cual tiene que estimar tres características antes de comenzar: cuánto durará, cuánto esfuerzo requerirá y cuántas personas estarán implicadas; es por eso que se tiene que facilitar un marco de trabajo que permita hacer estimaciones razonables. Donde se deben obtener estimaciones de esfuerzo humano requerido, así como la duración cronológica de dicho esfuerzo, la duración cronológica del proyecto y el costo, para garantizar un resultado más efectivo, seguro y real, para lograr una adecuada planificación productiva. Según **Pressman** "No podemos pedir exactitud a la fase de planificación, es solo una idea de cómo van a transcurrir las cosas. Hay que planificar el trabajo, los recursos humanos y la tecnología. Planificar es estimar". [1]

La estimación es importante no solo para predecir el valor de variables concretas dentro de un proyecto sino también para ver si se cuenta con el tiempo, el esfuerzo o los recursos necesarios para llevarlo a cabo y de esta forma determinar si es viable o no. Actualmente dicho proceso resulta muy engorroso, pues en muchas ocasiones se realiza teniendo un conocimiento mínimo de dicho proyecto y si no se cuenta con información real que pueda apoyar la estimación actual, no se pueden obtener estimaciones confiables y completas. El proceso de estimar nunca será una ciencia exacta, pero la combinación de buenos datos y técnicas puede mejorar la precisión de este. [2]

La Universidad de las Ciencias Informáticas (UCI), es un centro de enseñanza superior, surgido en el fragor de la Batalla de Ideas y obra del pensamiento visionario del comandante Fidel. Es la más joven de las universidades cubanas y por ello deudora del resto de las instituciones del país que le aportaron experiencias y personal. La misma juega un papel fundamental en el desarrollo de soluciones informáticas, convirtiéndose en una de las mayores productoras del país. Posee una estructura organizativa dividida en centros de software que están vinculados a la producción e investigación, tal es el caso del Centro de Tecnología de Gestión de Datos (DATEC), el cual tiene como misión facilitar soluciones integrales relacionadas con tecnologías de bases de datos y el análisis de información. Dicho centro desarrolla un conjunto de proyectos informáticos para la producción sobre la base de un modelo de Líneas de Productos de Software (LPS) que ayuda al desarrollo de la tecnología, reduciendo en tiempo y elevando la calidad de sus productos.

Actualmente se cuenta con una herramienta web creada para ayudar a los proyectos en la estimación de tiempo, esfuerzo y costo. Dicha herramienta está sustentada en el Método de Estimación UCI propuesto

por el Centro de Calidad (CALISOFT). El método es aplicado en la Fase de Estudio Preliminar del proyecto y requiere que sean introducidos en la herramienta manualmente los datos necesarios para el cálculo de las diferentes métricas. Es necesario un esfuerzo adicional para garantizar su actualización, tornándose engorrosa dicha tarea, cuando los datos pueden ser obtenidos directamente de los diagramas realizados en la herramienta "*Visual Paradigm for UML"*, donde puede detallarse una vista lógica de alto nivel que incorpore los sistemas, subsistemas, módulos y las características técnicas de la solución, lo que ahorraría tiempo y esfuerzo durante la elaboración del proyecto técnico. Tomando en cuenta lo anterior se tiene como **problema de la investigación**: ¿Cómo facilitar la estimación de proyecto a partir de los diagramas realizados en la herramienta "*Visual Paradigm for UML"*? Para resolver dicho problema se determina como **objeto de estudio**: el proceso de estimación de proyectos de software, enmarcado en el **campo de acción:** herramientas de estimación. El **objetivo general** de la presente investigación es: Desarrollar un *plugin* para "*Visual Paradigm for UML"* que permita realizar la estimación de proyecto aplicando el Método de Estimación UCI.

Para cumplir el objetivo trazado, se llevaron a cabo los siguientes objetivos específicos:

- Caracterizar el método de estimación UCI.
- Elaborar un perfil UML para reflejar información necesaria para realizar la estimación.
- Realizar el análisis, diseño e implementación del *plugin* de "*Visual Paradigm for UML".*
- Realizar pruebas funcionales al *plugin* implementado.

Para darle cumplimiento a los objetivos específicos del presente trabajo de diploma se definen las siguientes **tareas de la investigación:**

- Revisión bibliográfica de las herramientas existentes y de los enfoques teóricos relacionados con la estimación de proyectos fundamentalmente el Método de Estimación UCI.
- Valoración de las herramientas y tecnologías a utilizar en la construcción de la solución.
- Especificación de un perfil UML a aplicar en el modelado.
- Identificación de los requisitos funcionales y no funcionales.
- Realización del análisis.
- Realización del diseño.
- Implementación del *plugin.*
- Realización y documentación de las pruebas funcionales.

Para el desarrollo de la presente investigación se tendrán en cuenta los siguientes **métodos de la investigación** científica.

Dentro de los **teóricos** se emplearon los siguientes:

- **Analítico – Sintético**: este método es utilizado para, a partir de la investigación realizada acerca del Método de Estimación UCI y las herramientas de desarrollo de software existentes, definir las tecnologías y metodologías que serán utilizadas y arribar a las conclusiones de la investigación.
- **Histórico-Lógico:** este método permite desarrollar el estudio del arte, previo al desarrollo de la investigación, acerca de las estimaciones principalmente en el Método de Estimación UCI.

Dentro de los **empíricos** se empleó el siguiente:

- **Observación:** para determinar requisitos funcionales y características que debe tener el sistema.

Posible resultado esperado:

Se espera obtener un *plugin* que se integre a la herramienta CASE *Visual Paradigm for UML* que permita la estimación de proyecto a partir del Método Estimación UCI. Tal extensión beneficia el proceso de desarrollo de software en DATEC, particularmente al departamento de Integración de Soluciones, a partir de los artefactos resultantes de la ingeniería de requisitos, se realizara de manera automatizada la captura de las métricas necesarias para la estimación. Lo anterior puede resumirse concretamente en:

- *Plugin* de "*Visual Paradigm for UML*"capaz de realizar estimaciones de esfuerzo, tiempo y costos, a partir de un perfil de UML para reflejar métricas de estimación asociado al modelo.

Estructuración del trabajo de diploma:

Capítulo 1: Fundamentación teórica sobre los Métodos y Herramientas de Estimación de Proyecto de Software.

En este capítulo se analizan los elementos básicos a tener en cuenta para la implementación de *plugin* para la herramienta *"Visual Paradigm for UML".* Se realizará un análisis de las tecnologías y herramientas que servirán de apoyo a la investigación así como el análisis de las características y funcionalidades que permitirán definir las bases para la solución propuesta.

Capítulo 2: Análisis y Diseño del Plugin.

En este capítulo se define todo lo referente a las funcionalidades que debe tener el sistema para la realización del *plugin*. Se identifica el actor, casos de usos del sistema y la relación existente entre ellos, así como la vista lógica con el propósito de describir cómo se debe implementar el sistema enfatizando en las clases más relevantes. También se abordan los patrones arquitectónicos y de diseño presentes en la implementación del *plugin.*

Capítulo 3: Implementación y Pruebas del Plugin.

En este capítulo se presenta el modelo de implementación a través del diagrama de componentes. A demás se documentan las pruebas funcionales al *plugin*, para comprobar su correcto funcionamiento mediante los casos de uso.

1. Fundamentación teórica sobre los Métodos y Herramientas de Estimación de Proyecto de Software

1.1. Introducción

En este capítulo se estudia brevemente los conceptos básicos hacia la estimación para el desarrollo de software enfocándose en las bases del Método de Estimación UCI. Además se documenta la configuración del *plugin* para la herramienta *"Visual Paradigm for UML".* Se definen los elementos que conforman un perfil UML así como las herramientas, la metodología y las tecnologías para implementar el *plugin*.

1.2. Proceso de planificación

El proceso de planeación del proyecto depende de la planificación detallada de su avance, anticipando riesgos que puedan surgir y preparando soluciones tentativas a ellos, llevándose a cabo una estimación de los parámetros del proyecto como su estructura y tamaño. En los inicios del proyecto se requieren algunas estimaciones de costos antes que se tenga el cronograma detallado. Estas estimaciones son necesarias ya que establecen un presupuesto para el proyecto y así asignar un precio al software del cliente. Además permiten saber cuál es la fecha estimada de finalización del proyecto y en qué iteración estará lista una determinada funcionalidad. [3]

1.3. Métodos de estimación

Estimar es deducir, apreciar y dar valor. "Es una suposición cercana al valor real, normalmente, por medio de algún cálculo o razonamiento". Es prever con cierto grado de subjetividad el comportamiento de una variable en el futuro. En todas las áreas de las ciencias como en la vida cotidiana hay que estimar para obtener un valor aproximado de algo. Actualmente se estiman indicadores en el proceso de software: el tiempo, el costo y los recursos necesarios para su realización, para completar dicha actividad, es preciso recurrir a métodos aproximados. [2]

Existen diversos métodos de estimación, con características diferentes pero con objetivos comunes tales como:

- Establecer de antemano el alcance del proyecto.
- Usar como base para la realización de estimaciones, las métricas del software.
- Desglosar el proyecto en partes más pequeñas, estimando los costes y recursos de forma individual.

Inicialmente lo que se debe definir en un método de estimación son los indicadores a medir. Dicha medición es analizada para obtener la aproximación deseada. Se define indicador como: un valor

cuantitativo o cualitativo que expresa las características o estado de un individuo, objeto o proceso. Es una variable que permite conocer la magnitud o tamaño de algo, en relación con el total o un universo dado, que proporciona una visión profunda del proceso de desarrollo del software. [3]

1.3.1. Juicio de expertos

El método se basa en el análisis de un conjunto de opiniones y criterios que son emitidos por profesionales o expertos en una disciplina, relacionada con el proyecto que se está ejecutando. Un requerimiento antes de poner en práctica la estrategia, es tener bien definidas y descritas las tareas y actividades sobre las cuales los expertos deben emitir un juicio. Cuanto más específica sea la definición de una tarea o de una actividad, menos margen quedará para posibles interpretaciones erróneas. Por tanto, es preciso definir con claridad, el objetivo del ejercicio de evaluación, de manera tal que los expertos o jueces estén claros del trabajo que van a realizar.

1.3.2. Analogía

La estimación por analogía implica usar el costo real de proyectos anteriores, como base para estimar el precio del proyecto actual, se utiliza cuando la cantidad de información detallada sobre el proyecto es limitada por ejemplo, en las fases tempranas. Dicha estimación es menos costosa que otras técnicas pero también es menos exacta. Es más fiable cuando los proyectos anteriores son similares, no sólo en apariencia y las personas o grupos que preparan las estimaciones tienen la experiencia necesaria.

1.3.3. Métrica de puntos de función

Su propósito es medir el software cualificando la funcionalidad que proporciona, basándose en el diseño lógico del sistema. Los objetivos de puntos de función son:

- Medir lo que el usuario pide y lo que el usuario recibe.
- Medir independientemente la tecnología utilizada en la implantación del sistema.
- Proporcionar una métrica de tamaño con soporte al análisis de la calidad y la productividad.
- Proporcionar un medio para la estimación del software.
- Proporcionar un factor de normalización para la comparación de distintos software.

Este método calcula los puntos de función de un sistema descomponiéndolo en cinco funciones principales (entradas, salidas, consultas, ficheros internos y externos). Asignándoles valores de acuerdo a su complejidad y en función de la cantidad de cada uno de ellos se llega a determinar, mediante la sumatoria, los puntos de función, que son posteriormente ajustados de acuerdo con las características específicas del proyecto (International Function Point users Group).[4]

1.3.4.Métrica de puntos de casos de uso

Puntos de Casos de Uso es un método para la estimación del tamaño de proyectos basado en casos de uso. La técnica permite obtener estimaciones en etapas iníciales del proyecto, basta con tener un cierto entendimiento sobre el dominio del problema (plasmado en casos de uso) para comenzar a estimar. Informalmente, los casos de uso son historias sobre el uso de algún sistema con algún objetivo a alcanzar. La dificultad radica en descubrir los requerimientos y la escritura de los mismos con un nivel de detalle que sea útil). [4]

Este método de estimación y cálculo de tamaño del software, está basado en las cuentas hechas sobre los casos de uso para un sistema de software. El método exige la existencia de un modelo de casos de uso, por lo que la labor deberá ser hecha cuando exista algún entendimiento del dominio del problema o cuando se esté realizando las labores de arquitectura y dimensionamiento del tamaño del sistema.

1.3.5.Modelo COCOMO II

Es uno de los modelos de estimación de costo mejor documentado, estudiado y utilizado en la industria de software. “Barry Boehm en su libro "Economía de la ingeniería de software" detalla un modelo amplio de estimación de costos llamado COCOMO II (Constructive Cost Model).” La palabra *"constructive"* se refiere al hecho de que el modelo ayuda a un estimador a comprender mejor la complejidad del software; este modelo es un ejemplo de variable simple, estática y es usado por miles de administradores de proyecto de software. El modelo permite, basándose en un grupo de ecuaciones no lineales obtenidas mediante técnicas de regresión a través de un histórico de proyectos ya realizados; estimar el esfuerzo, costo y tiempo que se requiere en un proyecto de software a partir de una medida del tamaño del mismo, expresada en el número de líneas de código que se estimen generar para la creación del software. ”El modelo COCOMO original se ha convertido en uno de los modelos de estimación de coste del software más utilizados y estudiados en la industria. El modelo original ha evolucionado a un modelo de estimación más completo llamado COCOMO II. [4]

1.3.6. Método de Estimación UCI

En el marco del Programa de Mejora de la Universidad de las Ciencias Informáticas (UCI), se inició la concepción de un repositorio que permitiera almacenar y analizar los indicadores de la producción de software obtenidos a partir de los proyectos productivos. Un elemento fundamental de esta guía es proveer una infraestructura y procedimientos a seguir para colectar y obtener automáticamente dos tipos de indicadores: productividad y esfuerzo. Aun así, la estrategia que se describe en el propio documento del Repositorio Central para la UCI, incluye aún técnicas de recolección manual a través de plantillas lo

cual incide negativamente por el tiempo que demoraría la entrada de todos los datos del proyecto o cuando se hace algún cambio en el mismo. También la recolección manual incide como pérdida en la posterior obtención de indicadores a niveles organizacionales pues habría que hacer revisión de cada plantilla generada para poder extraer las cifras totales. Por tanto, conocer qué medidas, han de recopilarse en los proyectos de una Organización del Software, tanto genéricas como específicas, resulta una información imprescindible a la hora de emplear herramientas de gestión que soliciten dicha información a múltiples proyectos para ser almacenadas en Bases de Datos de Medición. [5]

1.4. Bases del Método de Estimación UCI para el desarrollo de software

El desarrollo de este método parte de la necesidad en la UCI de estimar el tiempo y esfuerzo requeridos para desarrollar un producto software.

1.4.1. Factores de complejidad que influyen en las métricas

Los factores de complejidad definidos se listan con un peso determinado en una escala de 0.5 - 2 en dependencia del nivel de importancia. La complejidad se proporciona en un primer momento como valoración cualitativa de cada factor y luego internamente se cuantifica en:

Escala de 0-5.

Evaluación	Valoración
Alto (A)	5
Medio Alto (MA)	4
Medio (M)	3
Medio Bajo (MB)	2
Bajo (B)	1

Tabla 1: Escala de 0-5.

Escala de 0-1.

Evaluación	Valoración
Si	1
No	0

Tabla 2: Escala de 0-1.

Luego de clasificar la complejidad por cada factor se procede a calcular el resultado final para cada uno al multiplicar el peso predeterminado para cada factor y el valor equivalente que toma cada valoración cualitativa.

El resultado final será la sumatoria del valor calculado para todos los criterios llevándose a una escala de conveniencia en función de su uso en la fórmula final del método. Los máximos se determinan a partir de configuraciones totalmente positivas o negativas.

Factor Cliente

El factor cliente permite determinar un nivel de incertidumbre asociado a la madurez que tiene el cliente en materia de asimilación de proyectos informáticos. Se utilizará para determinar la cantidad de unidades de desarrollo que se dejan como reserva.

No.	Factor	Descripción	Peso	Valoración	P*V
FC 1	Existe sistema anterior.	¿El proyecto se ha automatizado en otro momento?	0.5	(Si, NO)	
FC 2	Existen resultados de informatización.	¿Existen algunas funcionalidades o paquetes funcionales informatizados?	1	(Si, NO)	
FC 3	Existe Dirección de Informatización o Informática.	¿Se cuenta en la organización con personal calificado en los conocimientos informáticos y/o con una Dirección o grupo dedicado a la Informatización?	1	(Si, NO)	
FC 4	Existe infraestructura tecnológica en la organización.	La infraestructura tecnológica representa la integración de un conjunto de elementos de hardware (servidores, puestos de trabajo, redes, enlaces de telecomunicaciones, etc.), software (sistemas operativos, bases de datos, lenguajes de programación,	1	(Si, NO)	

		herramientas de administración, etc.) y servicios (soporte técnico, seguros, comunicaciones, etc.) que en conjunto dan soporte a las aplicaciones (sistemas informáticos) de una empresa u organización.			
FC 5	Existe base legal en la organización.	¿La organización cuenta con una base fundamentada jurídicamente y apoyada en la ley?	1	(Si, NO)	
FC 6	Es el primer proyecto con la organización.	¿Es la primera vez que se realiza un proyecto con la organización?	0.5	(Si, NO)	
FC 7	Existe una estructura clara en la organización.	¿Los procesos y áreas funcionales están claramente definidos?	1	(Si, NO)	
FC 8	Existe un especialista para atender al proyecto.	¿En la organización se cuenta con un especialista en disposición de atender el proyecto?	1	(Si, NO)	
FC 9	Están definidas las funciones de las áreas.	¿Las actividades que realizan en cada área están definidas?	1.5	(Si, NO)	
FC 10	Estabilidad de los requisitos.	¿Los requisitos funcionales y no funcionales se mantendrán estables y sin posibilidad de cambios?	1	(Si, NO)	
FC 11	El cliente es el	¿El cliente que	0.5	(Si, NO)	FC 11

	usuario de la aplicación.	solicita la realización del proyecto es el usuario que interactuará con la aplicación?			

Tabla 3: Factor de Cliente.

El Factor Cliente se calcula como: uno más, la sumatoria del producto de cada factor individual por su respectivo peso convertido a una escala entre 1 y 1.5 mediante la fórmula:

FC = FO/FP

Donde:

FC: Factor Cliente

FO: Factor obtenido

FP: Factor obtenido asignando valoraciones pesimistas.

Factor de Valor Agregado

El factor de valor agregado determina algunas acciones que pueden representar esfuerzo adicional al desarrollo de software, se incluye en todas las etapas, fundamentalmente en la de Análisis y Diseño. Se debe evitar seleccionar estas opciones en aquellos casos donde el nivel de desarrollo sea alto.

Factor	**Descripción**	**Valoración**	**P*V**
Ayuda Integrada.	La aplicación debe contar con una ayuda integrada al sistema.	(Si, NO)	
Mantener una interfaz familiar al negocio.	El cliente de la aplicación solicita que la interfaz o el diseño gráfico del sistema cuenten con características particulares o explícitas.	(Si, NO)	
Aplicación multidioma.	La aplicación debe ser desarrollada en varios idiomas.	(Si, NO)	

Tabla 4: Factor de Valor Agregado.

El Factor Valor Agregado se calcula como: uno más, la sumatoria del producto de cada factor individual por su respectivo peso convertido a una escala entre 1 y 1.5 mediante la fórmula:

VA = 1+FO*0.2/FP

Donde:

VA: Valor Agregado

FO: Factor obtenido

FP: Factor obtenido asignando valoraciones pesimistas

Factor Ambiente

Factor de Ambiente (FA): es interno, se determina a través de analizar las condiciones de la estructura de producción en la que se desarrollará el proyecto, no influye en el precio final, pero si en las valoraciones para la realización del proyecto y se utilizará como indicador para saber si el estructura de producción ha creado condiciones para cumplir los compromisos. Se aplicará en el precio de negociación y previo a la contratación.

Al obtener las valoraciones de las complejidades de los factores desde el FA1- FA8 se procede a contabilizar cuántos factores están considerados positivos y negativos (SI, NO):

- Si el total de valoraciones de complejidad negativas (NO) es 2 o menos, se utiliza el factor de conversión para el tiempo de 25 horas- hombre en la implementación de un punto de función.
- Si el total de valoraciones de complejidad negativas (NO) es igual o mayor que 4, se utiliza el factor de conversión para el tiempo de 30 horas- hombre en la implementación de un punto de función.

Si el FA8 es valorado de negativo (NO) entonces el tiempo de trabajo diario que se toma será de 5 horas como tiempo dedicado a trabajo diario en el peor de los casos donde se tienen involucrados trabajando a medio tiempo.

No.	Factor	Descripción	Peso	Valoración	P*V
FA 1	Temática conocida por la estructura de producción.	¿La estructura de producción tiene experiencia en el desarrollo de proyectos sobre esta temática o cuenta con personal capacitado en esta temática?	1.5	(Si, No)	
FA 2	La estructura de producción tiene conocimientos sobre Base de	¿La estructura de producción domina los conocimientos de Base de	1	(Si, NO)	

	Datos.	Datos? Las Bases de Datos son sistemas formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.			
FA 3	La estructura de producción tiene conocimientos sobre Leguaje de Programación.	¿La estructura de producción domina los conocimientos sobre el lenguaje de programación que especifica el cliente o con el que se pretende realizar el proyecto?	1	(Si, NO)	
FA 4	La estructura de producción tiene conocimientos sobre el Control de Versiones.	¿La estructura de producción domina los conocimientos y herramientas para el control de versiones? El control de versiones es la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una	1	(Si, NO)	

		configuración del mismo.			
FA 4	La estructura de producción tiene conocimientos sobre el Control de Versiones.	¿La estructura de producción domina los conocimientos y herramientas para el control de versiones? El control de versiones es la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo.	1	(Si, No)	
FA 5	La estructura de producción tiene conocimientos sobre la Gestión de Proyectos.	¿La estructura de producción tiene experiencia en el desarrollo de proyectos exitosos? La Gestión de proyectos es la disciplina de organizar y administrar recursos de manera tal que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, , el tiempo, y coste definidos.	1	(Si, No)	
FA 6	Proyecto de continuidad.	Se ha desarrollado una parte del	2	(Si, No)	

		proyecto y se pretende continuar el desarrollo.			
FA 7	Disponibilidad de la estructura de producción de líder de Proyecto.	¿La estructura de producción cuenta con personal con experiencia en la gestión de proyectos?	1	(Si, No)	
FA 8	La estructura de producción cuenta con todos los desarrolladores dedicados a tiempo completo en el proyecto	¿Los desarrolladores involucrados (estudiantes y profesores) solo están dedicados a la producción a tiempo completo?	1	(Si, No)	FA 8

Tabla 5: Factor Ambiente.

Factor de Complejidad Técnica

El factor de complejidad técnica es asociado a la tecnología y requerimientos no funcionales del sistema, influye directamente en el esfuerzo y fundamentalmente en la etapa de implementación.

No.	Factor	Descripción	Peso	Complejidad	P*V
CT 1	Rendimiento de la aplicación.	Velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia de la aplicación.	1	(A, MA, M, MB, B)	Peso*Valoración
CT 2	Rendimiento de la Plataforma.	Velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento	1	(A, MA, M, MB, B)	

		efectivo total y eficacia de la Plataforma.			
CT 3	Complejidad de Diseño Gráfico	Nivel de complejidad en el conjunto de operaciones técnicas-proyectuales necesarias para la información visual, al objeto de dotarla de la mayor cantidad posible de atributos eficaces, comprensibles y persuasivos para la fácil y completa percepción del mensaje a transmitir.	1	(A, MA, M, MB, B)	
CT 4	Procesamiento interno complejo.	Procesamientos lógicos o matemáticos intensivos en la aplicación	1	(A, MA, M, MB, B)	
CT 5	Incluye objetivos especiales de Seguridad.	La disponibilidad de mecanismos que controlen o protejan los programas o los datos.	1	(A, MA, M, MB, B)	
CT 6	Reutilización.	Definida por una métrica específica	0.5	Según Fórmula	
CT 7	Integración.	Definida por una métrica específica	1	Según Fórmula	
CT 8	Asimilación de Sistemas.	Definida por una métrica específica	1	Según Fórmula	

CT 9	Asimilación de Dispositivos.	Definida por una métrica específica	0.5	Según Fórmula	
CT 10	Asimilación de Estándares.	Definida por una métrica específica	1	Según Fórmula	

Tabla 6: Factor de Complejidad Técnica.

En el caso específico de este factor algunos de los valores para calcular la complejidad están dados por las siguientes métricas, para la cual existe una escala para llevarla a la ecuación cualitativa:

Porciento de Reutilización = CMR/CMD

- Cantidad de Módulos con Reutilización (CMR).
- Cantidad de Módulos a Desarrollar (CMD).

Porciento de Integración = CSI / CMS

- Cantidad de Sistemas a Integrar o Legar (CSI).
- Cantidad de Módulos del Sistema (CMS).

Porciento de Asimilación de Sistemas = CSI-D / CSI

- Cantidad de Sistemas a Integrar o Legar (CSI).
- Cantidad de Sistemas a Integrar o Legar Desconocidos (CSI-D).

Porciento de Asimilación de Dispositivos = (CDE – DRV) / CDE

- Cantidad de Dispositivos Externos (CDE).
- Cantidad de Dispositivos Externos con Driver o SDK (DRV).

Porciento de Asimilación de Estándares = CEE / CEC

- Cantidad de Estándares y Normas a cumplir (CEC).
- Cantidad de Estándares y Normas específicas (CEE).

El valor del factor para cada caso será la proporción entre las medidas sin ser convertido a valores porcentuales, por lo que para estos factores no aplican los criterios cuantitativos.

El factor de complejidad técnica se calcula como: uno más, la sumatoria del producto de cada factor individual por su respectivo peso convertido a una escala entre 1 y 1.5 mediante la fórmula:

FC = 1+FO*0.5/FP

Donde:

FO: factor obtenido

FP: factor obtenido asignando valoraciones pesimistas.

1.4.2.Métricas definidas en el Método de Estimación UCI

El primer paso del método es la identificación de la cantidad de Paquetes Funcionales (PF) que potencialmente tendrá el sistema a desarrollar. Estos PF engloban una serie de funcionalidades y estarán compuestos por una determinada cantidad de puntos de función (pf) que será estimada por el método. Estos PF se clasificarán según la complejidad atendiendo al tamaño.

Cantidad de Puntos de Función Máximos y Mínimos según clasificación del Paquete Funcional		
Clasificación	**Cota Mínima**	**Cota Máxima**
Pequeño	3	10
Mediano	11	18
Grande	19	26

Tabla 7: Cantidad de Puntos de Función según la clasificación de los Paquetes Funcionales.

Métricas de Tamaño

Para unificar los puntos de función según la clasificación de los paquetes funcionales, se aplicará una métrica donde influye el Factor Cliente (FC). Quedando:

- TPFG = TPFG *CMaxG *FC donde,
  TPFG Tamaño Paquetes Funcionales Grandes (cantidad de puntos de función)
  TPFG: Total de Paquetes Funcionales Grandes.
  CMaxG: Cota Máxima de puntos de función para un paquete funcional Grande.
- TPFM = TPFM *CMaxM *FC donde,
  TPFM: Tamaño Paquetes Funcionales Medianos (cantidad de puntos de función)
  TPFM: Total de Paquetes Funcionales Medianos.
  CMaxM: Cota Máxima de puntos de función para un paquete funcional Mediano.
- TPFP = TPFP *CMaxP *FC donde,
  TPFP: Tamaño Paquetes Funcionales Pequeños (cantidad de puntos de función)
  TPF: Total de Paquetes Funcionales Pequeños.
  TPFP: Total de Paquetes Funcionales Pequeños.

CMaxP: Cota Máxima de puntos de función para un paquete funcional Pequeño.

Obtenido el valor del Tamaño para cada tipo de PF se procede a calcular el tamaño total:

- UDA= Σ (TPFG+ TPFM + TPFP)*CT

Métricas de Tiempo

Para determinar el tiempo de desarrollo por cada etapa identificada se realizó un análisis de lo propuesto por el método de Puntos de Casos de Uso ajustándolo a la UCI. Los ajustes fueron validados y consultados por algunos expertos, definiéndose los porcientos que representan el total de los tiempos dedicados a cada actividad.

Porciento de tiempo por actividad respecto al total de tiempo de desarrollo	
Estudio Preliminar	6.06%
Modelación del Negocio	13.14%
Captura de Requisitos	19.18%
Análisis y diseño	17.12%
Implementación	31.32%
Pruebas internas	10.10%
Pruebas de Liberación	3.08%

Tabla 8: Porciento de tiempo por actividades estándares de desarrollo.

Para calcular tiempo total de desarrollo se inicia por el cálculo del tiempo de la fase de Implementación (TImp), teniendo en cuenta que el tiempo más fácil de estimar es el de implementación de un Punto de Función.

TImp = TImpUnitario * PFA

TImpUnitario: Constante que identifica el Tiempo de Implementación de 1 punto de función.

TImpUnitario = 25 horas a tiempo completo y 30 horas a medio tiempo dedicado a la implementación.

Después se calcula el tiempo de Análisis y Diseño (TAD) utilizando los porcientos definidos para cada fase, agregándole el factor de valor agregado (FAG) multiplicado por la cantidad de PFA.

TAD = (TImp * %AD)/%Imp)* FAG.

Donde:

%AD: % de tiempo correspondiente a la fase de Análisis y Diseño.

El tiempo dedicado a la captura de requisitos, las pruebas internas y otras actividades se mantienen calculándose sobre la base del tiempo de Implementación (TImp).

Tiempo dedicado a la Estudio Preliminar (TEP)= (TImp * % Estudio Preliminar)/ % TImp

- Tiempo dedicado a la Modelación del Negocio (TMN)= (TImp * % Modelación del Negocio)/ % TImp.
- Tiempo dedicado a la Captura de Requisitos (TCR)= (TImp * % Captura de Requisitos)/ % TImp.
- Tiempo dedicado a las pruebas internas (TPI)= (TImp * % Pruebas Internas)/ % TImp.
- Tiempo dedicado a las Pruebas de Liberación (TPL)= (TImp * % Pruebas de Liberación)/ % TImp.

El Tiempo Real de Desarrollo (TRD) se calcula utilizando el Esfuerzo de desarrollo (E) y el Total de Hombres (TH).

- TRD = E/TH

Métrica del esfuerzo

Para calcular el esfuerzo de desarrollo se tendrán en cuenta todos los tiempos dedicados a las etapas.

- E= Σ(TEP+ TMN+TCR+ TAD+ TImp+ TPI+ TPL)

Para calcular el Esfuerzo máximo de desarrollo (EMax) se tendrán en cuenta el tiempo dedicado a trabajar diariamente en el proyecto. Que en el peor de los casos cuando hay involucrados a tiempo medio como promedio es de 5 horas y en el mejor de los casos cuando la vinculación a la producción es a tiempo completo como promedio es de 8 horas. La cantidad de días laborables a la semana que son 5 y el TLS de proyecto.

- EMax= HD*DL*TLS :

Donde:

HD: Tiempo de trabajo diario.

DL: Cantidad de días laborables a la semana.

TLS: Tiempo Límite se Solución.

Métricas de recursos

El total de hombres (TH) para el desarrollando se obtendrá a partir del esfuerzo de desarrollo y el esfuerzo máximo de desarrollo el tiempo real de desarrollo (TRD).

- TH= E/TRD

Métricas de Costo

Para calcular el costo de desarrollo de software (CS) se necesita partir del tiempo que llevará el desarrollo y las tarifas asociadas a la cantidad de roles clasificados en el proyecto.

Tarifas por rol		
Rol	**Mínimo (TMín)**	**Máximo(TMax)**
Técnico	4.47	6.25
Profesional	6.58	10.41
Consultor de Alto Nivel	11.05	15.62

Tabla 9: Tarifas por rol.

El costo del software se calcula a partir del TRD:

- CS= (TRD * totalConsultores * TarifaConsultores) + (TRD * totaltécnicos * Tarifatécnicos) + (TRD * totalProfesinales * TarifaProfesionales)

Donde totalConsultores, totaltécnicos y totalProfesinales se calculan a partir del % que representa cada rol en el total de miembros del equipo.

El Método de Estimación UCI ha considerado características de otros métodos el cual es aplicado en los proyectos de desarrollo de software, con el objetivo de estimar el tamaño, costo y esfuerzo requerido para desarrollar un producto de software. Con este se logra minimizar el grado de incertidumbre de los atributos de estimación y permiten establecer un presupuesto a los productos de desarrollo de software. Se tiene en cuenta una serie de factores de complejidad que influyen en las métricas para estimar los atributos, el tiempo dedicado a cada fase del desarrollo del software y la tarifa por rol. Esta ajustado sobre la base de la experiencia que se tiene en la universidad.

1.5. Elementos arquitectónicos para las extensiones a "*Visual Paradigm for UML".*

La implementación de extensiones para aplicaciones constituye un mecanismo para la incorporación de funcionalidades que la aplicación no provee a los usuarios. Por lo general se asocian las extensiones con *plugin,* que son fragmentos de software que interactúan con el núcleo de la aplicación para proporcionar algunas funcionalidades que en la mayoría de los casos son muy específicos. *"Visual Paradigm for UML"* es una de las herramientas *CASE* que goza de gran prestigio para el modelado de software y cuenta la librería openapi.jar para extender funcionalidades. La aplicación provee de forma libre una interfaz de

programación (API por su siglas en inglés Interfaz de Programación de Aplicaciones) permitiendo a los desarrolladores implementar y reutilizar clases e interfaces, desarrollando funciones agregadas que son útiles para el desarrollo de software.

¿Cómo se implementa un *plugin* para la herramienta *"Visual Paradigm for UML"*?

Para la implementación de un *plugin* en "*Visual Paradigm for UML*" es importante conocer la estructura de desarrollo, así como la de integración con la herramienta. Para la consecución de las acciones se debe realizar como:

La herramienta *"Visual Paradigm for UML"* provee el mecanismo de extensión a través de un openapi.jar. Es una librería que está ubicada en el paquete de instalación de la herramienta, dentro del paquete "lib/openapi.jar". Una vez conocida cual es la librería de construcción de *plugin* es necesario tener en cuenta la estructura que conforma el *plugin* para su desarrollo. Para el IDE Netbeans tiene la estructura correspondiente a un paquete que lleva el nombre de plugin el cual contiene los paquetes asociados, a la configuración del *plugin*, a las acciones correspondientes y a los formularios o diálogos de la implementación. De ellas es importante dominar su funcionamiento y la relación que se establece por medio del openapi.jar incorporado al proyecto.

Estructura de la implementación del *plugin Estimavp*:

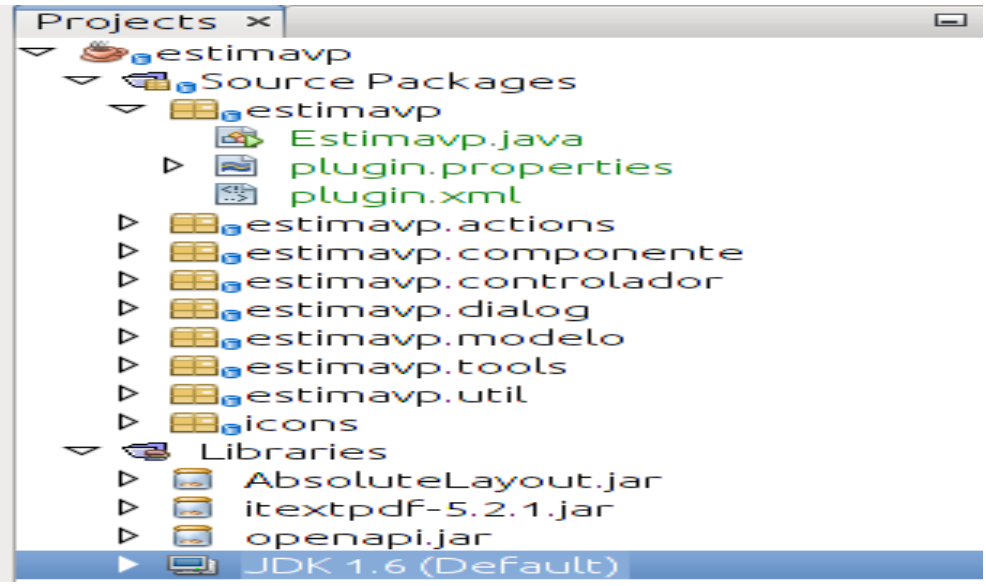

Ilustración 1: Estructura del proyecto *Estimavp* para "*Visual Paradigm for UML*" en IDE NetBeans.

1.6. Lenguaje Unificado de Modelado (UML) 2.0

El UML es el lenguaje estándar especificado por el *Object Management Group* (OMG por sus siglas en inglés) para visualizar, especificar, construir y documentar los artefactos de un sistema, y además sirve para el modelado del negocio y sistemas de software.[6] Este ofrece un estándar para describir los modelos, incluyendo aspectos conceptuales como procesos de negocio, funciones del sistema, expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

UML cuenta con un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. UML es una notación, por lo que no puede ser considerado un método.[6]

Algunas de las principales ventajas que tiene la utilización de UML en la construcción de sistemas de software son:

- Puede usarse en las diferentes etapas del ciclo de vida del desarrollo de sistemas.
- Es independiente del proceso o metodología de desarrollo y del lenguaje de implementación.
- Permite crear y modificar modelos expresivos mediante la combinación de los 13 tipos de diagramas que suministra en su versión 2.0.
- Es posible extender la funcionalidad de la notación gráfica mediante estereotipos y proveer una base formal para los diagramas.

La nueva versión de UML 2.0 mejora de forma más clara las relaciones permitidas entre los elementos del modelo a especificar y el uso de las metaclases de un metamodelo dentro un perfil UML.

1.7. Perfil de UML

Actualmente UML es el estándar más utilizado por la industria del software para realizar el modelado de sistemas, pero a pesar de todas las ventajas que ofrece no posee la capacidad de detallar todo tipo de dominio, en especial aquellos que presentan un alto nivel de detalle. Sin embargo, UML cuenta con algunos mecanismos de extensión, entre los que se encuentran los perfiles, que permiten llevar a cabo el modelado de sistemas con un nivel superior de detalle.

Los perfiles UML forman parte del estándar de UML definido por la OMG para extender metamodelos de referencia existentes, con el fin de adaptarlos a un determinado dominio o plataforma. Su uso permite disponer de una terminología propia del dominio de la aplicación objetivo y definir una nueva notación para símbolos ya existentes más acorde con este. Un perfil UML permite añadir restricciones a un metamodelo además de las ya existentes e incorporar información que puede ser útil a la hora de transformar el modelo a otros metamodelos o a código fuente.[7]

Los perfiles UML están compuestos por un conjunto de elementos relacionados entre sí, ellos son las clases, extensiones, perfiles, perfiles de aplicación, paquetes y estereotipos. Actualmente no existe un estándar definido para la construcción de perfiles UML, pero algunos autores han propuesto una secuencia de pasos a tener en cuenta para ello: [7]

- Definir el dominio de aplicación a modelar con el perfil.
- Definir el perfil.
- Definir cuáles son los elementos de UML del dominio que se están extendiendo sobre los que es posible aplicar un estereotipo.
- Definir los valores etiquetados de los elementos del perfil.
- Definir las restricciones que forman parte del perfil, a partir de las restricciones del dominio.

Para desarrollar la solución propuesta en la investigación, fue necesario definir un perfil de UML para proveer a "*Visual Paradigm for UML*" de un conjunto de funcionalidades que permiten reflejar las informaciones de las métricas para mostrar el reporte de estimación.

1.8. Metodología para el desarrollo de software

1.8.1. Open Unified Process OpenUP/Basic

Para llevar a cabo la implementación del *plugin* se empleó como metodología de desarrollo OpenUp/Basic en consideración con las políticas del centro DATEC para el desarrollo de software. Además está pensado para equipos pequeños. OpenUp/Basic es desarrollo iterativo e incremental dentro de la estructura del ciclo de vida del software que es mínimo, completo y extensible. El proceso es mínimo cuando solamente el contenido principal es incluido; es completo cuando puede ser mostrado todo el proceso para construir un sistema y extensible en el que puede ser utilizado como fundamento sobre el cual el contenido de proceso se pueda agregar o adaptar según lo necesitado. Es una metodología ligera que puede ser utilizada en varios tipos de proyectos de software, beneficiando a clientes y desarrolladores sobre productos a entregar y su formalidad. Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo. [8]

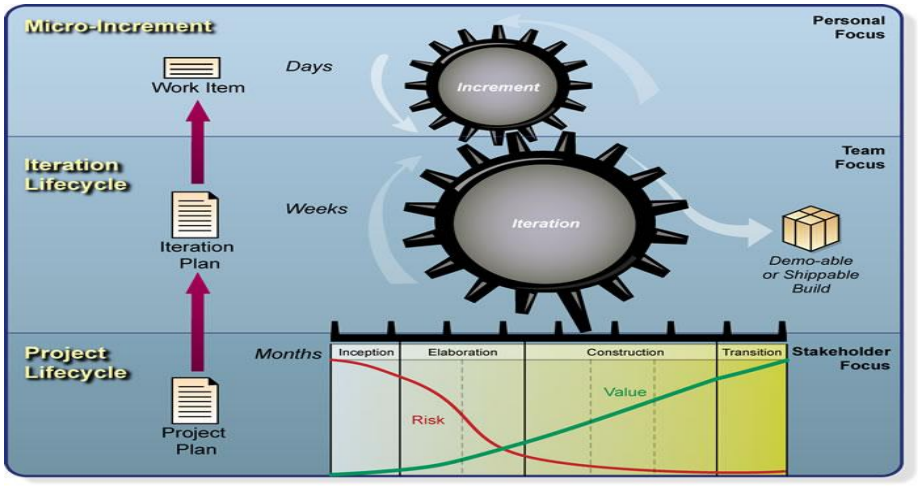

Ilustración 2: Capas de la metodología “OpenUP”.

1.9. Lenguaje de programación

Un lenguaje de programación describe un conjunto de acciones que un equipo debe ejecutar. Está conformado por un conjunto de símbolos, reglas sintácticas, semánticas que definen su estructura, el significado de sus elementos y expresiones. Al conjunto de instrucciones que se genera se conoce como código fuente de un programa. 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.

1.9.1. Lenguaje de programación Java

Para la implementación del *plugin* se empleó como lenguaje de programación Java, por ser el lenguaje propuesto por el API de desarrollo de la herramienta *“Visual Paradigm for UML“.* Java es un lenguaje de programación orientado a objeto, de plataforma independiente. El lenguaje toma sintaxis de lenguajes como C y C++, aunque tiene un modelo de objeto más simple y elimina herramientas de bajo nivel. Su característica distintiva lo ha llevado a niveles muy alto en la preferencia de los desarrolladores de la comunidad internacional. [9]

1.10. Herramientas a utilizar

1.10.1. Visual Paradigm for UML 8.0

Para la modelación del *plugin*, se utilizó como herramienta “*Visual Paradigm for UML”* por ser la escogida por la universidad para el desarrollo de software y asumida por el centro DATEC. Soporta el ciclo de vida completo en el desarrollo de software: análisis y desarrollos orientados a objetos, construcción, prueba y despliegue. También brinda un diseño centrado en casos de uso y enfocado al negocio lo que genera un

software de mayor calidad. Permite dibujar todo tipo de diagrama de clases, código inverso, generación de código a partir de diagramas y generar documentación. Posee una disponibilidad de múltiples versiones, además de integrarse al IDE de desarrollo NetBeans que es el utilizado para la implementación del *plugin.* [10]

1.10.2. IDE NetBeans 6.9

Para el desarrollo de la implementación del *plugin* se utilizó como entorno de desarrollo integrado (IDE), NetBeans 6.9, que constituye un entorno de desarrollo modular y basado en estándares, escrito con el lenguaje de programación Java. Es una aplicación de código abierto *("open source")* diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas. Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. Es un producto libre y gratuito sin restricciones de uso. [11]

1.11. Conclusiones parciales

Se estudió el Método de Estimación UCI como punto de partida para la implementación del *plugin*, las tecnologías, las herramientas y la metodología que serán utilizadas en el desarrollo para el *plugin*. Teniendo en cuenta las exigencias y necesidades del centro DATEC, se decidió utilizar OpenUP/Basic como metodología de desarrollo de software y como lenguaje de modelado UML 2.0, el cual brinda soporte para visualizar, construir y documentar los artefactos del sistema. Se emplea Java como lenguaje de programación, como herramientas "*Visual Paradigm for UML"* en su versión 8.0 para el modelado del *plugin*, NetBeans 7.1 como IDE de programación.

2. Análisis y Diseño del Plugin

2.1. Introducción

En el presente capítulo se describen los conceptos más importantes en el entorno del sistema. Se manifiestan los requerimientos funcionales y no funcionales que se deben tener en cuenta para la implementación del *plugin.* También, se identifica el actor, casos de usos y las relaciones existentes entre ellos. Se aborda acerca de cómo debe funcionar el sistema y se hace una descripción general de las actividades. Se muestra además, los diagramas de clases del diseño, así como los diagramas de secuencia correspondientes que serán objeto de automatización.

2.2. Propuesta de solución

Se propone realizar un *plugin* para la herramienta "*Visual Paradigm for UML"* que implemente el Método de Estimación UCI. Dicho método consta de métricas y factores los cuales tienen que ser calculados y procesados por el *plugin* del "*Visual Paradigm for UML"*. El elemento fundamental es cómo se obtienen estas métricas y estos factores de los modelos del "*Visual Paradigm for UML"*; *es* necesario definir un perfil UML, que permita reflejar dentro los modelos la información de estas métricas y estos factores, para ser interpretado por el *plugin* y de esta forma mostrar el reporte de estimación.

2.3. Especificación del perfil UML

Los perfiles UML forman parte del estándar de UML definido por la OMG para extender metamodelos de referencia existentes con el fin de adaptarlo a un determinado dominio o plataforma, permiten disponer de una terminología propia del dominio de la aplicación objetivo y definir una nueva notación para símbolos ya existentes más acorde con este y constituyen el mecanismo que proporciona el propio UML para extender su sintaxis y su semántica para expresar los conceptos específicos de un determinado dominio de aplicación, es un lenguaje que proporciona una gran flexibilidad y expresividad a la hora de modelar sistemas.[12]

Los modelos proporcionan un mayor nivel de abstracción, permitiendo trabajar con sistemas mayores y más complejos, y facilitando el proceso de codificación e implementación del sistema de forma distribuida y en distintas plataformas. [12]

La especificación del perfil UML está compuesta por un conjunto de estereotipos que heredan de sus metaclases correspondientes, como se muestra a continuación:

Ilustración 3: Especificación del perfil UML.

En la siguiente especificación del perfil UML para el modelo de domino se establecen las entidades sistemas legados, paquetes funcionales y las unidades de desarrollo que constituyen estereotipos. Cada estereotipo extiende de una metaclase, lo que permite que cuando se especifique un elemento de ese mismo estereotipo se asegure que se empleen las mismas reglas semánticas de la metaclase a la cual extiende. Los estereotipos utilizados son *module* y *legacy.* El estereotipo *module* extiende a la metaclase *Package* y *legacy* extiende a la metaclase Actor.

2.3.1. Especificación de los estereotipos del perfil de UML

A continuación, son mostrados y descritos los estereotipos que componen al perfil de UML definido.

- **Estereotipo:<<module>>**

Metaclase a la que extiende: *Package.*

Semántica del estereotipo: el estereotipo es aplicable a paquetes que agrupan a casos de uso dentro del marco de un sistema. Un módulo es una agrupación lógica de unidades funcionales estrechamente relacionadas que, en su conjunto, encierran un mayor valor agregado para el usuario del sistema.

- **Estereotipo:<<legacy>>**

Metaclase a la que extiende: Actor.

Semántica del estereotipo: este estereotipo representa a los actores que son externos al sistema.

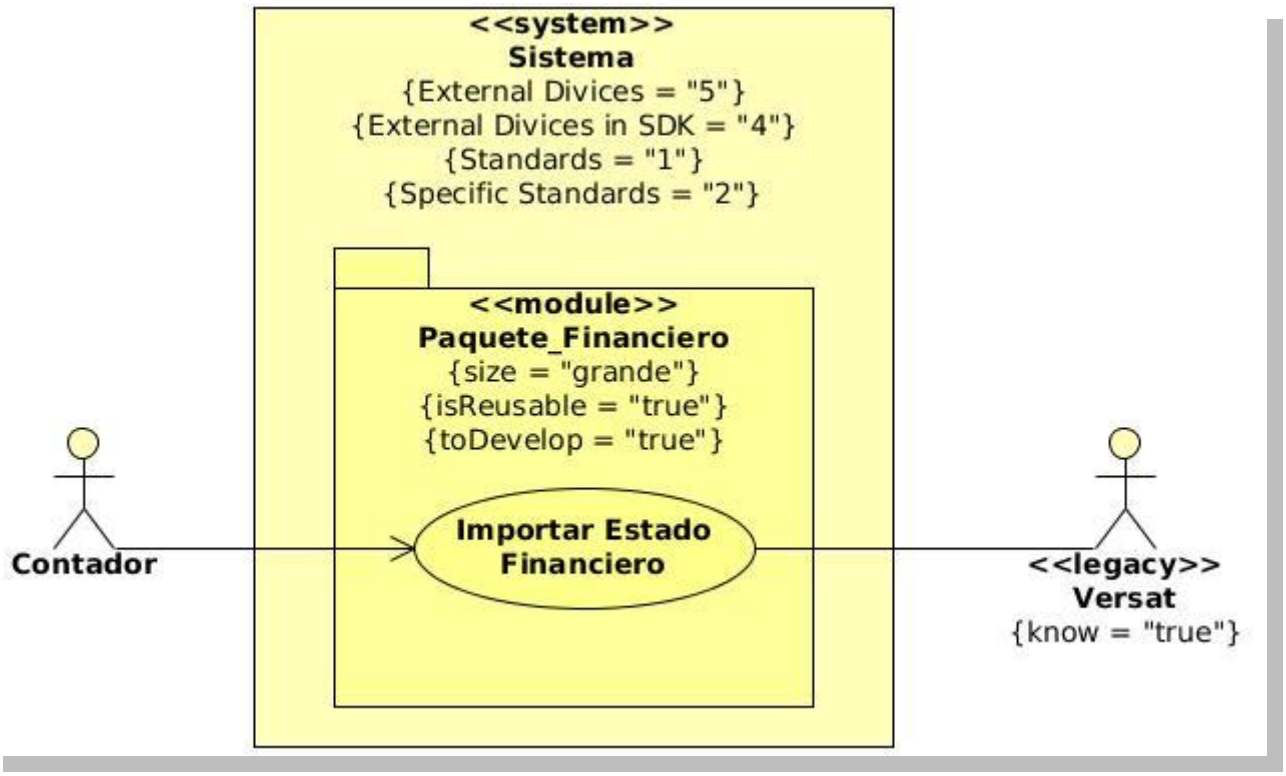

Ilustración 4: Ejemplo del perfil UML.

2.4. Modelo de dominio

Un modelo del dominio captura los tipos de objetos más importantes que existen, o los eventos que suceden en el entorno donde estará el sistema, se identifican y se definen los conceptos relacionándose en un diagrama de clases UML. El objetivo fundamental de este modelo es comprender y describir los conceptos más importantes dentro del contexto del sistema. El modelo de dominio es una representación visual del entorno real del proyecto. [13]

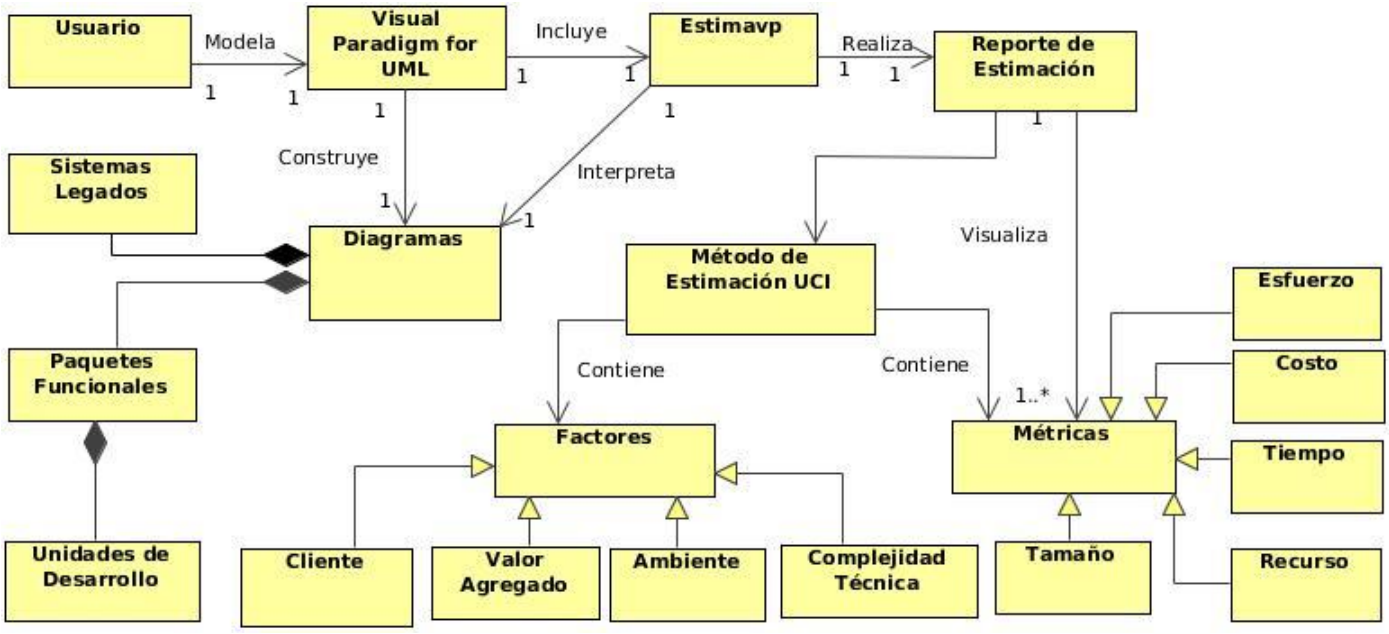

Ilustración 5: Modelo del dominio.

2.4.1. Definición de las clases del modelo del dominio

- **Usuario:** es el responsable de modelar en la herramienta "*Visual Paradigm for UML".*
- **Visual Paradigm:** es la herramienta de modelado que permite crear diversos modelos entre los que se encuentra el perfil UML que representa el punto de partida para calcular la estimación.
- **Diagramas:** son representados por las clases y están relacionados entre sí, que contienen atributos y operaciones.
- **Sistemas legados:** es la cantidad de módulo a integrar que hay en el sistema.
- **Paquetes funcionales:** son los módulos o paquetes que forman parte del producto a desarrollar.
- **Unidades de desarrollo:** es la unidad de medida estándar para el tamaño de los proyectos que están guiados por los casos de uso.
- **Estimavp:** es el nombre del *plugin* y representa la aplicación que se integra con "*Visual Paradigm for UML"* para realizar la estimación.
- **Reporte de estimación:** es la lista de los resultados de la estimación generada en formato PDF por el *plugin.*

- **Método de Estimación UCI:** permite obtener información necesaria para realizar la estimación se realiza en la herramienta "*Visual Paradigm for UML"*. A demás se tuvo en cuenta la evaluación de algunos factores y métricas que influyen en el método para obtener estimación del proyecto.
- **Métricas:** ayudan a entender el proceso del método, que se utiliza para desarrollar un *plugin.*
- **Tamaño:** es la métrica que unifica las unidades de desarrollo según la clasificación de los paquetes funcionales, donde influye el factor cliente.
- **Recurso:** es la métrica que calcula el total de hombres necesarios para el desarrollo del software, que se obtendrá a partir del esfuerzo de desarrollo y el esfuerzo máximo de desarrollo del tiempo real de desarrollo.
- **Tiempo:** es la métrica que calcula porciento de tiempo por actividad respecto al total de tiempo de desarrollo.
- **Costo:** es la métrica que calcula el costo de desarrollo de software que se necesita partir del tiempo que llevará el desarrollo y las tarifas asociadas a la cantidad de roles clasificados en el proyecto.
- **Esfuerzo:** es la métrica que calcula el esfuerzo máximo de desarrollo se tendrán en cuenta el tiempo dedicado a trabajar diariamente en el proyecto.
- **Factores:** estos influyen en el proceso del método que se utiliza para calcular la estimación.
- **Cliente:** es un factor que determina la cantidad de unidades de desarrollo.
- **Valor agregado:** es un factor que establece algunas acciones que pueden representar esfuerzo adicional al desarrollo de software.
- **Ambiente:** es un factor que analiza las condiciones de la estructura de producción en la que se desarrollará el sistema.
- **Complejidad técnica**: es el factor asociado a los requerimientos no funcionales del sistema, influye directamente en el esfuerzo y fundamentalmente en la etapa de implementación.

2.5. Especificación de los requisitos

2.5.1. Requisitos funcionales

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Los requisitos funcionales definen las funciones que el sistema será capaz de realizar. Expresan la naturaleza del funcionamiento del sistema cómo interacciona el sistema con su entorno y cuáles van a ser su estado y funcionamiento. [14]

A continuación se muestran los requisitos funcionales:

RF1. Identificar los paquetes funcionales.

Identificación de la cantidad de paquetes funcionales (pf) que potencialmente tendrá el sistema a desarrollar.

RF2. Clasificar las unidades de desarrollo.

Clasificación de las unidades de medidas para el tamaño de los proyectos según los paquetes funcionales guiados por casos de uso.

RF3. Calcular métricas de tamaño.

Definir las métricas de tamaño según la clasificación de los paquetes funcionales.

RF4. Calcular métricas de tiempo.

Determinar el tiempo de desarrollo por cada etapa.

RF5. Calcular métricas de recurso.

Se obtendrá a partir del esfuerzo de desarrollo y el esfuerzo máximo de desarrollo el tiempo real de desarrollo.

RF6. Calcular métricas de costo.

Calcular el costo de desarrollo de software que se necesita a partir del tiempo que llevará el desarrollo y las tarifas asociadas a la cantidad de roles clasificados.

RF7. Calcular factores de complejidad.

Se define un peso determinado en una escala de 0.5 - 1.5 en dependencia del nivel de importancia. La complejidad se da en un primer momento como una valoración cualitativa de cada factor y luego internamente se cuantifica en una evaluación y una valoración.

RF8. Calcular factor cliente.

Determinar un nivel de incertidumbre asociado a la madurez que tiene el cliente en materia de asimilación de proyectos informáticos.

RF9. Calcular valor agregado.

Se determina algunas acciones que pueden representar esfuerzo adicional al desarrollo de software.

RF10. Calcular factor ambiente.

Se determina a través de analizar las condiciones de la estructura de producción en la que se desarrollará el proyecto.

RF11. Calcular factor de complejidad técnica.

Es asociado a la complejidad de la tecnología y requerimientos no funcionales del sistema, influye directamente en el esfuerzo y fundamentalmente en las etapas de implementación.

RF12: Consultar estimación

Para consultar la estimación de haberse realizado la estimación del algún proyecto.

RF13: Generar reporte de estimación en formato PDF.

Después de haberse calculado la estimación se generan los datos en formato de PDF.

2.5.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general los requisitos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener. [14]

Con el objetivo de que la aplicación posea una mayor calidad se proponen una serie de requisitos no funcionales, que le brindarán mayor eficacia, a la vez que la harán más confiable, rápida y atractiva para los usuarios.

Se especifican los siguientes requisitos no funcionales asumiendo los de la herramienta ““*Visual Paradigm for UML”*, ya que el *plugin* por sí mismo no cumple funcionalidad:

Requisitos de Software

- Se debe tener instalada la herramienta “*Visual Paradigm for UML”*. en su versión 8.0.
- Se debe tener el JRE (Java Runtime Environment).

Requisitos de Hardware

- La capacidad mínima para la memoria RAM (Random Access Memory, por sus siglas en inglés) debe ser de 512 MB pero se recomienda 1,0 GB.
- La capacidad mínima de espacio en disco debe ser 800 MB.

Restricciones del diseño y la implementación

- Se hace uso de la herramienta “*Visual Paradigm for UML”*. en su versión 8.0 y IDE NetBeans 7.1.
- El lenguaje de programación que será usado para la implementación es Java.

Requisitos de Usabilidad

- Permitir la accesibilidad a las operaciones, en caso de no tener mouse.

Interfaz

- Debe tener una interfaz amigable y con apariencia profesional.
- La interfaz debe tener un diseño sencillo y ser de fácil comprensión para el usuario.

2.6. Modelo de casos de uso del sistema

El modelo de casos de uso del sistema representa las relaciones existentes entre actores y casos de uso. Un modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema, representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un caso de uso es una unidad de trabajo significativo. Cada caso de uso tiene una descripción que especifica la funcionalidad que se incorporará al sistema propuesto. Un caso de uso puede 'incluir' la funcionalidad de otro caso de uso o puede 'extender' a otro, con su propio comportamiento. [15]

2.6.1. Actores del sistema

Un actor es un usuario del sistema, que usa un caso de uso para desempeñar alguna porción de trabajo que es de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define su rol global en el sistema y el alcance de su acción. Además son generalmente responsables de realizar actividades que serán automatizadas en el futuro sistema. [15]

Nombre del Actor	Descripción
Planificador	Es el encargado de interactuar con la herramienta "*Visual Paradigm for UML"*., para realizar la estimación y después consultarla.

Tabla 10: Descripción del actor del sistema.

2.6.2. Diagramas de casos de uso

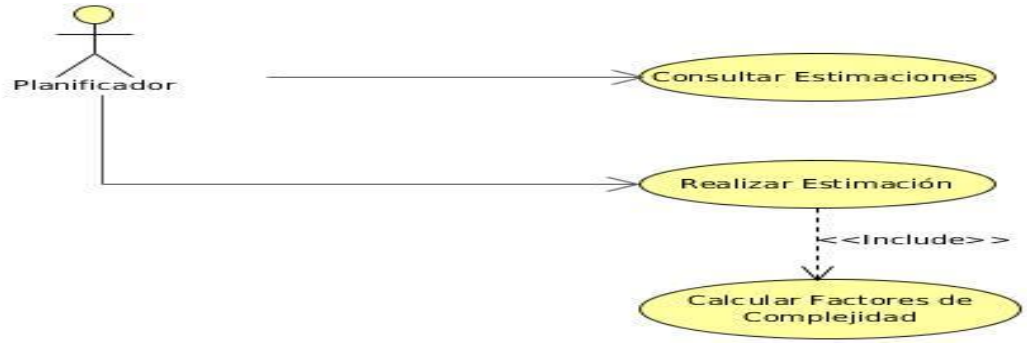

Ilustración 6: Diagrama de casos de uso del sistema.

2.6.3.Descripción textual de los casos de uso

Caso de Uso del Sistema: Realizar Estimación

Objetivo	Realizar Estimación	
Actores	Planificador	
Resumen	El caso de uso comienza al seleccionar la opción “Realizar Estimaciones, en la barra de herramienta del “*Visual Paradigm for UML”*, donde se muestra la interfaz realizar la estimación por el Método Estimación UCI con todos los campos que el usuario de llenar.	
Complejidad	Alta	
Prioridad	Crítico	
Precondiciones	Que exista un diagrama de casos de usos con las especificaciones realizadas en el Perfil UML diseñado.	
Postcondiciones	Se ha realizado la estimación a través del Perfil UML según las especificaciones realizadas en el perfil diseñado.	
Flujo de eventos		
Flujo básico “Realizar Estimación”		
	Actor	**Sistema**
1.	Selecciona la opción **“Realizar Estimaciones”,** en la barra de Herramienta del “*Visual Paradigm for UML”*.	Muestra la interfaz Realizar estimación por el Método Estimación UCI con todos los datos que el usuario debe llenar, como cantidad de roles, las actividades en semana. Así como los valores de los factores de complejidad. Ir a la sección 1.
2.	Introduce los datos, la cantidad de roles y las actividades en semanas.	Ir flujos alternos. Los datos son incorrectos.
3.	Selecciona la opción Realizar Estimación.	Muestra una interfaz con el resultado de la estimación de acuerdo con los parámetros entrados anteriormente.
4.	Selecciona la opción Cerrar.	El sistema cierra la interfaz correctamente. Terminando así el caso de uso.

Flujos alternos		
Nº Evento Los datos son incorrectos		
	Actor	**Sistema**
1.		El sistema muestra un mensaje, “los campos tienen que ser con valores enteros”.
2.		Si se deja campos vacios el sistema muestra un mensaje, los campos del formulario no pueden estar vacios.
Sección 1: “Calcular Factores de Complejidad”		
Flujo básico *“*Calcular Factores de Complejidad”		
	Actor	**Sistema**
1.		El sistema muestra los diferentes tipos de factores asociados, todos con su valor de complejidad.
2.	Selecciona los factores asociados.	Calcula los factores de complejidad siguiendo las fórmulas empleadas en el Método de Estimación UCI.
Relaciones	**CU Incluidos**	**Calcular Factores de Complejidad**
Requisitos no funcionales	**Software, restricciones del diseño y la implementación e interfaz.**	

Interfaz de Usuario

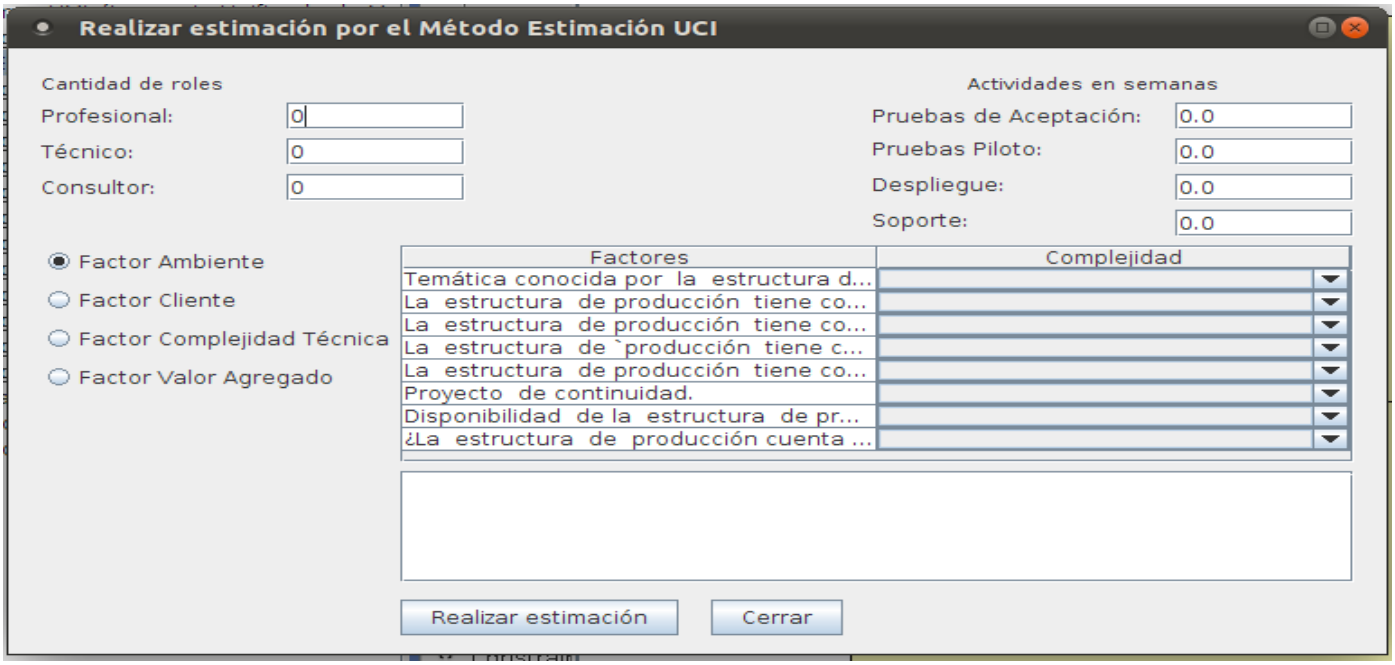

Tabla 11: Descripción textual del CU: Realizar Estimación.

Caso de Uso del Sistema: Consultar Estimaciones

Objetivo	Consultar Estimaciones
Actores	Planificador
Resumen	El caso de uso comienza al seleccionar la opción “Consultar Estimaciones. Muestra una ventana donde se genera un reporte con el resultado de las estimaciones realizadas.
Complejidad	Media
Prioridad	Crítico
Precondiciones	Se tiene que haber realizado la estimación del proyecto.
Postcondiciones	Se ha generado el reporte de estimación correctamente.
Flujo de eventos	
Flujo básico “Consultar Estimaciones”	

	Actor	Sistema
1	Selecciona la opción "**Consultar Estimaciones**", en la barra de herramienta del "*Visual Paradigm for UML".*	Muestra una interfaz con el resultado de la estimación realizada, las actividades y sus tiempos así como el tiempo máximo y mínimo de desarrollo del proyecto.
2	Selecciona la opción de Generar PDF.	Se levanta una ventana de diálogo donde da la posibilidad al usuario de seleccionar la dirección donde quiere guardar el reporte de la estimación.
3	Selecciona la opción Cerrar	El sistema cierra la ventana con los campos requeridos. Terminado así el caso de uso.
Requisitos no funcionales	**Software, restricciones del diseño y la implementación e interfaz.**	

Interfaz de Usuario

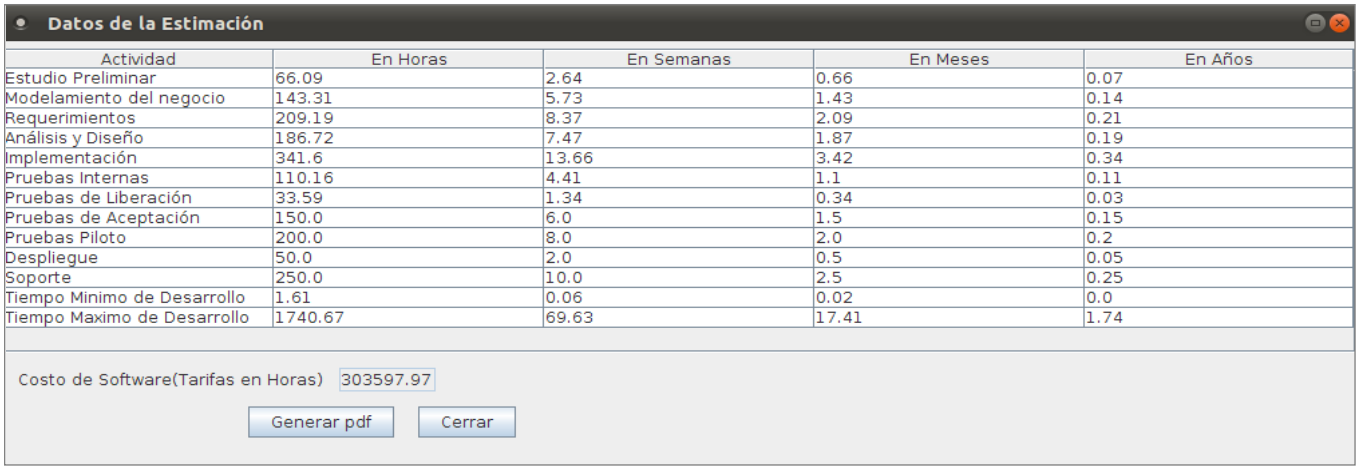

Tabla 12: Descripción textual del CU: Consultar Estimaciones.

2.6.4. Matriz de trazabilidad

La matriz de trazabilidad es una técnica que relaciona los requisitos funcionales a los diferentes elementos del desarrollo, permitiendo determinar qué requisitos quedan cubiertos por casos de uso. De esta forma garantizar que todos los elementos para el desarrollo del *plugin* sean ejecutados correctamente.

Requisitos Funcionales	CU1-Realizar Estimaciones	CU2-Calcular Factores de Complejidad	CU3-Consultar Estimaciones
RF1	X		
RF2	X		
RF3		X	
RF4		X	
RF5		X	
RF6		X	
RF7		X	
RF8		X	
RF9		X	
RF10		X	
RF11		X	
RF12			X
RF13			X

Tabla 13: Matriz de trazabilidad para los casos de uso.

Mediantes estos datos se puede saber o identificar qué requisitos funcionales están cubiertos por caso de uso y de esta forma garantizar que todos los elementos para el desarrollo del *plugin* sean ejecutados correctamente.

2.7. Modelo de diseño

Roger Pressman expresó en su libro **"Ingeniería de Software: Un Enfoque práctico "**que el diseño de software es un proceso de muchos pasos pero que se clasifican dentro de uno mismo". En general, la actividad del diseño se refiere al establecimiento de las estructuras de datos, la arquitectura general del

software, representaciones de interfaz y algoritmos. El proceso de diseño traduce requisitos en una representación de software.

El diseño intenta preservar la estructura definida en el modelo de análisis. Debe ser una guía que puedan leer, entender los que construyan el código, los que prueban y mantienen el software. Este modelo se puede utilizar para visualizar la implementación y para soportar las técnicas de programación gráfica de la aplicación.

El modelo de diseño es un modelo de objetos que describe la realización de casos de uso, y sirve como una abstracción del modelo de aplicación y su código fuente. El modelo de diseño se utiliza como parte esencial para las actividades en ejecución y prueba, que se basa en el análisis y los requisitos de la arquitectura del sistema. Representa los componentes de aplicación y determina su colocación adecuada y el uso dentro de la arquitectura en general del sistema. [16]

2.7.1.Diagrama de casos de uso del diseño

Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, los componentes que se encargaran del funcionamiento y la relación que existe entre ellos. A continuación se muestran los diagramas de clases del diseño los cuales se modelaron separados por CU.

Diagrama de Clases del Diseño - CU Realizar Estimación

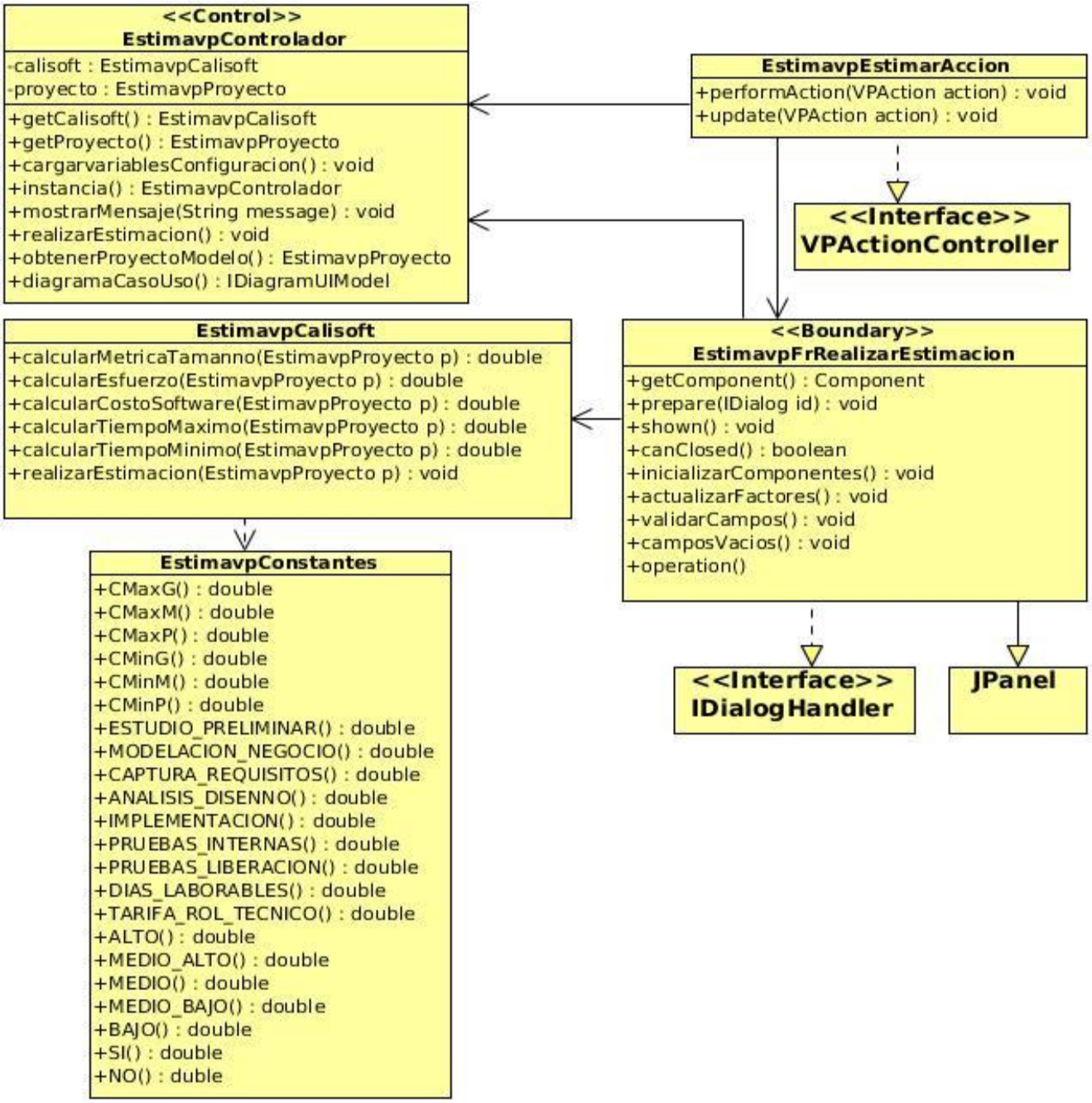

Ilustración 7: Diagrama de clases del diseño – CU Realizar Estimación.

Diagrama de Clases del Diseño - CU Consultar Estimaciones

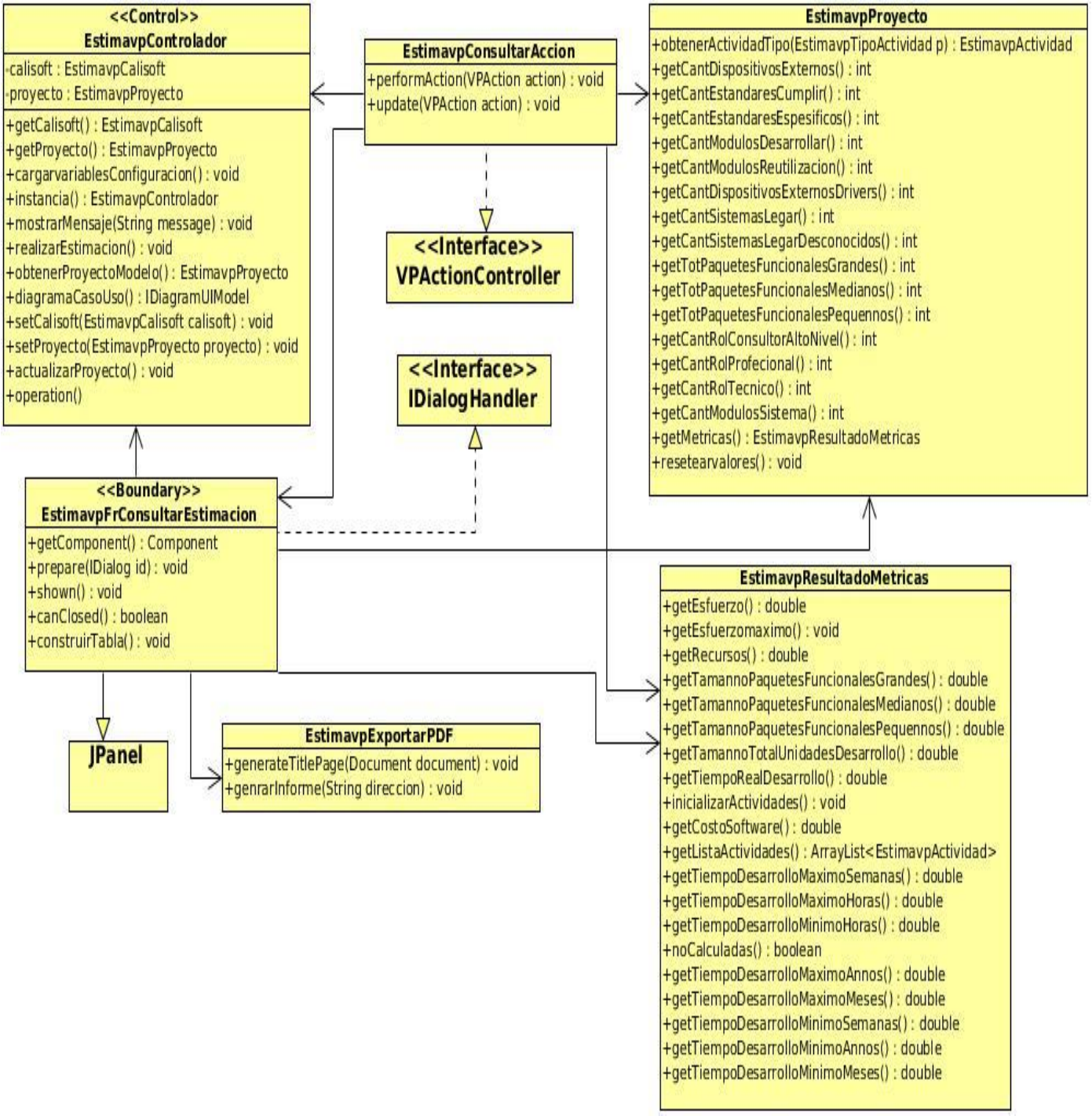

Ilustración 8: Diagrama de clases del diseño – CU Consultar Estimaciones.

2.7.2.Descripción de las clases más relevantes de los casos de uso del diseño

Diagrama de Clases del Diseño - CU Realizar Estimación

#	Clase	Tipo de clase	Descripción	Relación con otra clase	Dependencia
1	EstimavpControlador	Control	Es la clase controladora que implementa el patrón Singleton para poder utilizar una sola instancia de la clase controladora y el método realizar estimación, entre otras funcionalidades para el desarrollo del *plugin.*	3	Asociación
2	EstimavpFrRealizarEstimacion	Vista	Es la clase que contiene todos los elementos del formulario, además del control de todas las acciones que se realizan en la interfaz IDialogHandler.	1	Asociación
3	EstimavpEstimarAccion	Entidad	Es la clase referente al control de las acciones a nivel de herramientas, la misma implementa la interfaz VPActionController definida por el openapi.jar para el manejo de acciones para realizar la estimación.	1 y 2	Asociación

Tabla 14: Descripción de las clases relevantes del diseño para el CU: Realizar Estimación.

Diagrama de Clases del Diseño- CU Consultar Estimaciones

#	Clase	Tipo de clase	Descripción	Relación con otra clase	Dependencia
1	EstimavpConsultarAccion	Entidad	Es la clase maneja las acciones a nivel de herramientas, la misma implementa la interfaz VPActionController definida por el openapi.jar para integración con el "*Visual Paradigm for UML"*.	2	Realización
2	VPActionController	Interface	Es la clase que brinda el OpenAPI por defecto, la que permite actualizar cada acción realizada a través de la herramienta, así como ejecutar las acciones.	-	-
3	EstimavpFrConsultarEstimacion	Vista	Es la clase que representa el formulario asociado a la consulta de estimación, mostrando los datos generales del proyecto, generándolo a PDF.	1	Asociación

Tabla 15: Descripción de las clases relevantes del diseño para el CU: Consultar Estimaciones.

2.8. Diagrama de interacción

Los diagramas de interacción son utilizados para modelar los aspectos dinámicos de un sistema. En estos diagramas se representan un conjunto de objetos y sus relaciones incluyendo los mensajes que se pueden enviar entre ellos, no son más que una proyección de los elementos de una interacción.

Diagrama de Secuencia – Escenario: Realizar Estimación

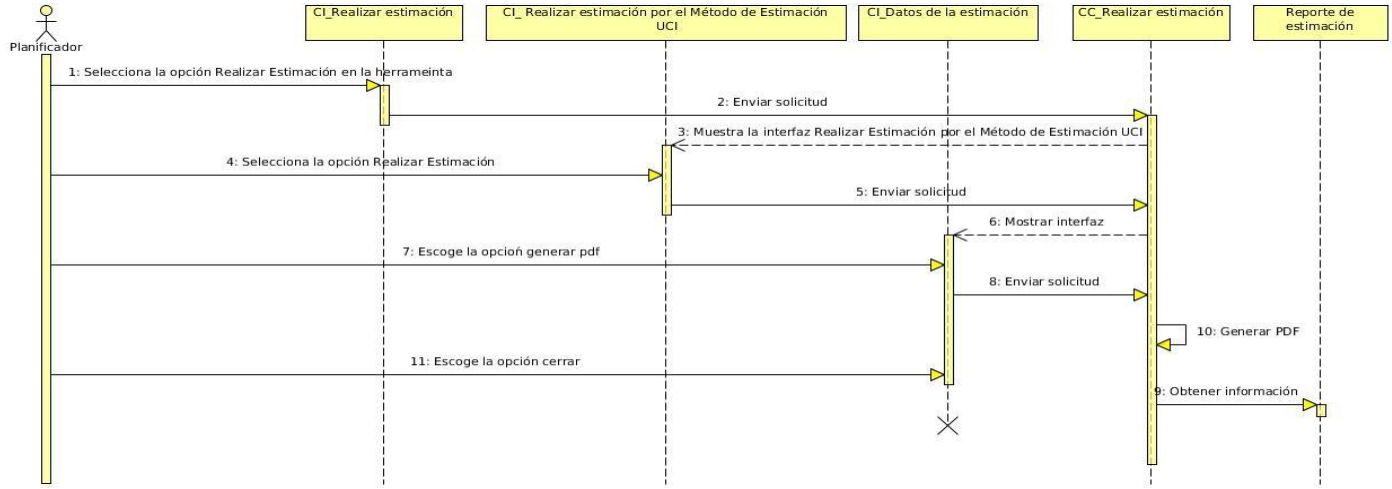

Ilustración 9: Diagrama de secuencia – Escenario: Realizar estimación.

Diagrama de Secuencia – Escenario: Consultar Estimaciones

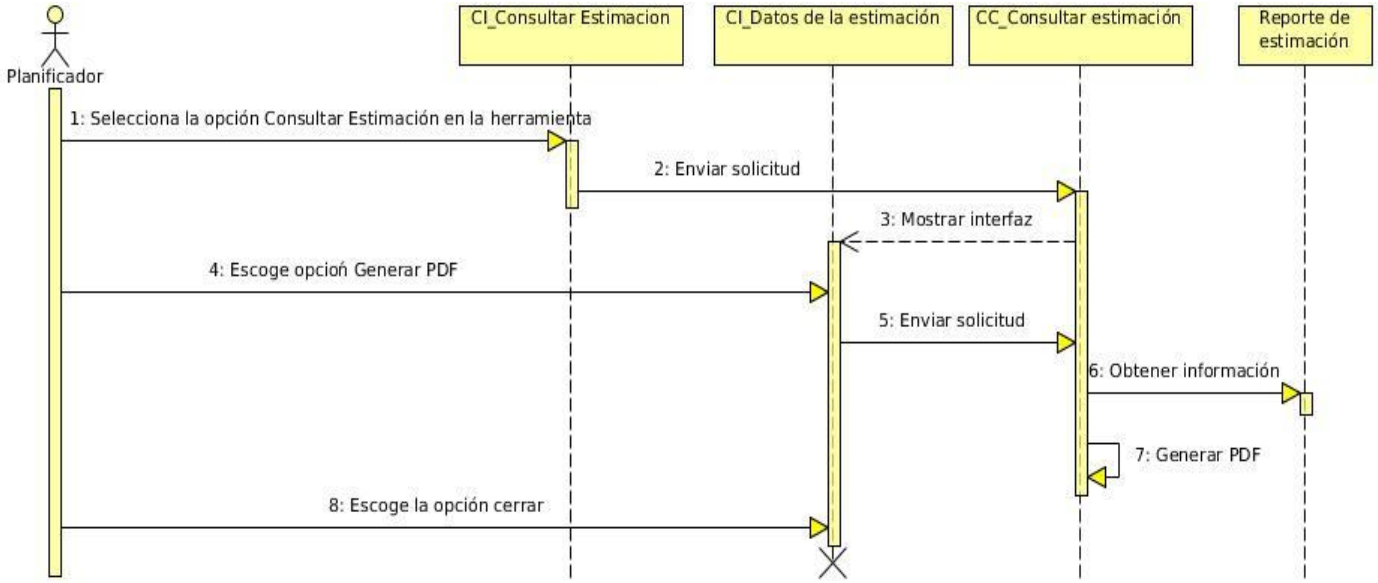

Ilustración 10: Diagrama de secuencia – Escenario: Consultar estimaciones.

2.9. Patrones utilizados

El uso de patrones constituye una base en la búsqueda de soluciones a problemas que surgen durante el desarrollo de software. Según el arquitecto Christopher Alexander[2], "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma. Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software". [17]

2.9.1. Patrón Arquitectónico: Modelo-Vista-Controlador

"*Visual Paradigm for UML"*. es la herramienta utilizada en el departamento Integración de Soluciones para el desarrollo de software. Esta provee una interfaz de programación de aplicaciones (API por sus siglas en inglés) que permite a los desarrolladores implementar y reutilizar clases e interfaces para desarrollar funciones agregadas de software.

Para definir la arquitectura de la extensión propuesta, resulta fundamental regirse por los elementos arquitectónicos definidos en el API de la herramienta. Esta propone una arquitectura basada en el patrón arquitectónico Modelo-Vista-Controlador (MVC), a través del cual es posible separar el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- **Modelo**: es el responsable de administrar el comportamiento del dominio de la aplicación, empleando para ello las acciones contenidas en el *openapi.jar*. Dichas acciones están asociadas a los elementos del modelo, y contemplan la creación de estereotipos, atributos, operaciones, clases y sus relaciones.
- **Vista:** es la responsable de la visualización de la información, a través de los diferentes tipos de diagramas que provee "*Visual Paradigm for UML"* y las interfaces mostradas al desarrollador.
- **Controlador:** es el responsable de interpretar los eventos producidos por el usuario, informando al modelo y/o a la vista para que cambien según resulte apropiado. Ejemplos representativos de esta clase son las acciones implementadas, las cuales capturan los elementos del modelo (Modelo) y a partir de ellos realizan acciones sobre los diagramas (Vista), y viceversa.

Este patrón fue utilizado ya que permite una separación de conceptos, de forma tal que el desarrollo de la aplicación esté estructurado de una mejor forma, también facilita la programación de manera paralela e independiente en las diferentes capas, además de brindarle una mayor escalabilidad.

2.9.2. Patrones de diseño GOF

Los patrones de diseño GOF son clasificados según su propósito en creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos.

- ***Factory Method* o Método de fabricación:**

El método de fabricación, es de tipo creación a nivel de clases que define una interfaz para crear un objeto, permitiendo a las subclases decidir de qué clase instanciarlo. Permite que una clase delegue en sus subclases la creación de objetos. Con este método se gana en flexibilidad, se facilita en cuanto a que se hace natural, la conexión entre jerarquías de clases paralelas, que son aquellas que se generan cuando una clase delega algunas de sus responsabilidades en una clase aparte. Ambas jerarquías de clases paralelas son creadas por un mismo cliente y el patrón método de fabricación establece la relación entre parejas de subclases. [18]

Este patrón resulta útil para la etapa de implementación del *plugin*, pues garantiza la instanciación de los objetos a través de métodos creacionales, que permiten la creación de los componentes y elementos de los modelos, garantizando el correcto funcionamiento del *plugin*. Su utilización se evidencia en la clase controladora *EstimavpControlador.java*, específicamente en el método *contabilizarElementosDiagrama(EstimavpProyecto p, IModelElement elementoModelo)*.

- ***Singleton* (Solitario)**

Es de tipo creacional, a nivel de objetos. Su propósito es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. El acceso a la "instancia única" es controlado. Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "solitario". Se utiliza cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes. [19]

El patrón *singleton* o solitario es implementado por la clase EstimavpControlador. java para poder utilizar una sola instancia de la clase controladora.

- ***Iterator* (Iterador)**

El patrón Iterador, es de tipo comportamiento a nivel de objetos que proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna. Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. Con la aplicación de este patrón se incrementa la flexibilidad, dado que para permitir nuevas formas de recorrer una estructura basta con modificar el iterador en uso, cambiarlo por otro o definir uno nuevo. Además se facilitan el paralelismo y la concurrencia, puesto que, como cada iterador tiene consciencia de su estado

en cada momento, es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente.

Se debe utilizar cuando se quiere acceder a los elementos de un objeto agregado sin mostrar su representación interna, cuando se quieren permitir recorridos múltiples en objetos agregados y cuando se quiera proporcionar una interfaz uniforme para recorrer diferentes estructuras de agregación. [19]

En la implementación del *plugin* es aplicado este patrón en clases como *EstimavpControlador.java* y *EstimavpProyecto.java*. En esta última clase se pone de manifiesto en el método *obtenerDatosModelo(),* iterando sobre los diagramas de casos de uso para obtener los paquetes del modelo.

2.9.3.Patrones diseño GRASP

Son patrones que resaltan la importancia de captar principios para la programación, a la hora de diseñar eficazmente un software orientado a objetos. Los patrones GRASP describen entre otros elementos la asignación de responsabilidades a objetos, expresados en forma de patrones.

- **Experto**

El uso del patrón experto permite asignar a una clase la información necesaria para cumplir su responsabilidad, de esta forma, las funcionalidades son asignadas de forma adecuada, facilitando la futura reutilización de componentes. Uno de sus beneficios consiste en permitir conservar el encapsulamiento, ya que los objetos se valen de su propia información para realizar las acciones que se les solicita.[20]

- **Creador:**

Este patrón asigna responsabilidades relacionadas con la creación de objetos y en la solución propuesta se utiliza al asignarle la responsabilidad a las controladoras de crear un objeto de las clases modelos para el posterior acceso a las funciones implementadas en el modelo. [20] Esto se evidencia dentro del marco de trabajo en las clases *EstimavpConsultarAccion.java* y *EstimavpEstimarAccion.java*. En dichas clases se encuentran implementadas varias acciones, dentro de las cuales se crean objetos de otras clases, lo cual evidencia que estas clases son creadoras de las que son instanciadas.

- **Controlador**

Es un patrón que sirve como intermediario entre una interfaz y el algoritmo que la implementa, de tal forma es la que recibe los datos del usuario y los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. [20] Se pone de manifiesto en la clase controladora *EstimavpControlador.*

- **Bajo Acoplamiento**

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. [20] En la implementación del *plugin* es aplicado este patrón en la mayoría de las clases, propiciando que los componentes sean fáciles de entender por separado y de reutilizar.

- **Alta Cohesión:**

Este patrón permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. [20] Ejemplos de ello son las clases *EstimavpFactorAbstracto, EstimavpFactor* y *EstimavpResultadoMetricas*, las cuales están formadas por varias funcionalidades relacionadas, siendo las responsables de definir las acciones y colaborar con otras para realizar diferentes operaciones, instanciar objetos y acceder a sus propiedades. Cada una de ellas contiene solamente operaciones correspondientes con su responsabilidad.

Para la implementación y funcionalidad del *plugin* debe implementarse patrones arquitectónicos, así como los patrones de diseño con el fin de agilizar el proceso de implementación.

2.10.Conclusiones parciales

Se confeccionó el modelo de dominio permitiendo una mejor compresión dentro del contexto del sistema. La identificación de los requisitos funcionales y no funcionales permitió definir las características y las condiciones que debe proveer el *plugin.* La especificación del perfil UML permitió reflejar la información en los diagramas que pueden ser modelados en la herramienta *"Visual Paradigm for UML".* Se describen los patrones de diseño empleados para el proceso de desarrollo del *plugin* que posibilitó asignar las responsabilidades adecuadas a cada una de las clases y el patrón arquitectónico MVC que permite una separación de conceptos, de forma tal que el desarrollo de la aplicación esté estructurado de una mejor forma.

3. Implementación y Pruebas del Plugin

3.1. Introducción

En el presente capítulo se muestra el modelo de implementación que pone en práctica el diseño de la solución que se va a realizar, así como el diagrama de componentes del plugin. Se describen las pruebas a realizar, con el objetivo de comprobar las funcionalidades del *plugin* en los diferentes escenarios, para de esta forma verificar en todos los casos que los resultados de las pruebas sean los esperados.

3.2. Modelo de implementación

El modelo de implementación describe cómo se implementan los elementos del modelo de diseño en términos de componentes. Además, describe los componentes a construir y su organización en nodos físicos en los que funcionará el sistema. [21]

Un diagrama de despliegue muestra las relaciones físicas entre los componentes de hardware y software en el sistema final, es decir la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software. Solo aquellos que son utilizados en tiempo de compilación deben mostrarse en el diagrama de componente, por lo antes expuesto se define: realizar el modelo de implementación mediante el diagrama de componente, para la descripción de los componentes del diseño ya que los componentes de un plugin son compilados por la herramienta *"Visual Paradigm for UML"* iniciada la aplicación.

3.2.1. Diagrama de componentes

El diagrama de componentes es usado para estructurar el modelo de implementación, describe los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Un componente representa un elemento físico que forma parte del sistema, se puede representar por paquetes y sus operaciones solo se pueden alcanzar a través de interfaces. [22]

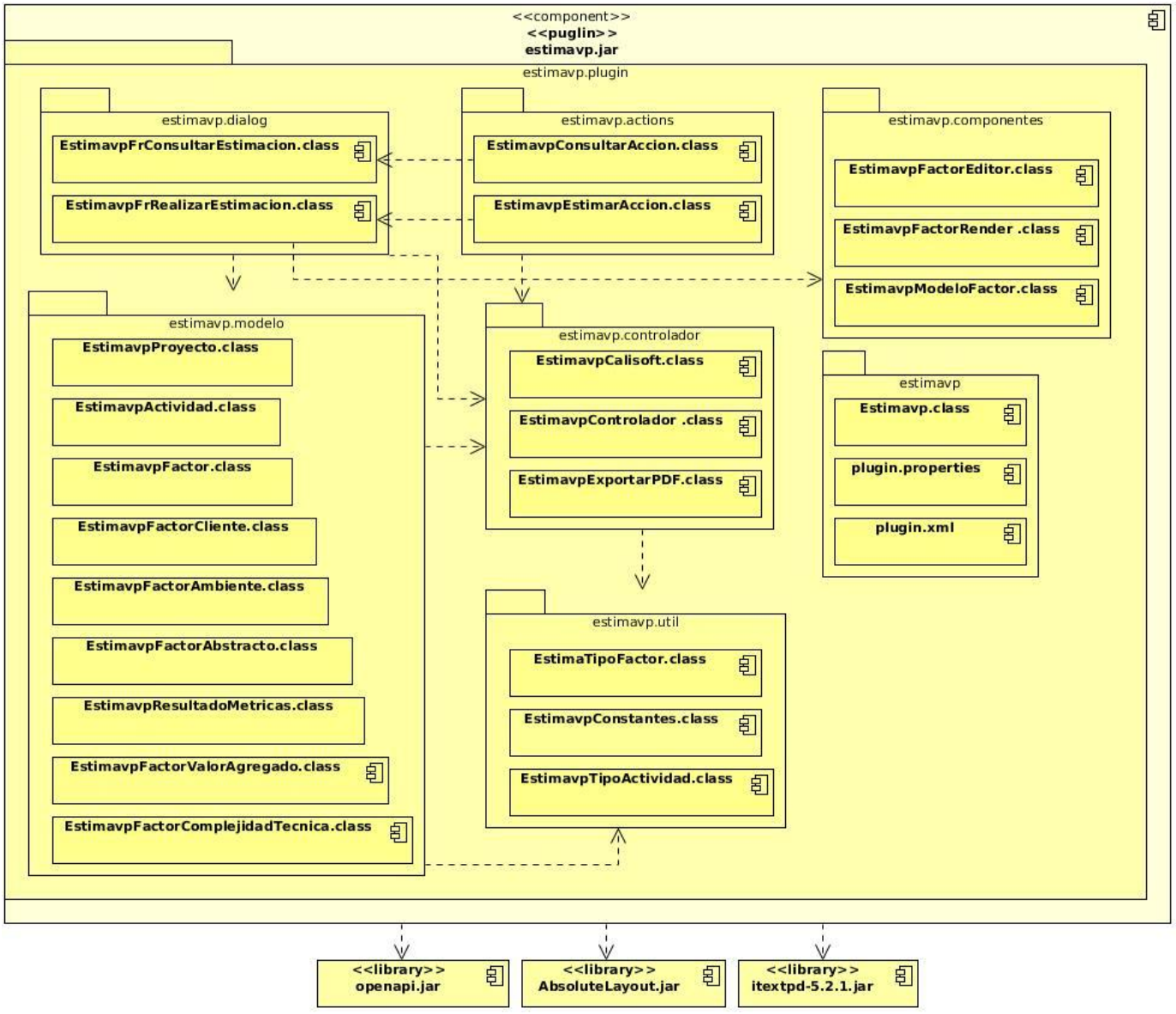

Ilustración 11: Diagrama de componentes.

3.2.2. Descripción de los componentes más relevantes

El presente modelo de implementación describe cada uno de los componentes asociados al diseño de clases propuesto en el desarrollo del plugin para la herramienta de modelado *"Visual Paradigm for UML"*, así como la relación de dependencia entre los componentes que la integran.

Nombre del plugin: "*estimavp"* representa la aplicación del modelado en *"Visual Paradigm for UML"* para realizar la estimación a partir del modelo del sistema.

Paquete estimavp.plugin: es el paquete de la estructura de un *plugin* en "*Visual Paradigm for UML"* contenedor de los paquetes asociados la configuración de plugin en la implementación.

Paquete estimavp: agrupa las clases asociadas a la configuración del *plugin* estas son: plugin.xml, Estimavp.java y plugin.properties definidas en la estructura que propone la herramienta para la extensión.

Paquete estimavp.actions: es el paquete contiene las acciones del *plugin* definidas a nivel de herramientas y de contexto. Dichas clases en dependencia de la acción, implementa interfaces asociadas a las acciones, VPActionController y VPContextAction. Ambas clases definen el performAction el cual permite ejecutar acciones referentes al evento onClick de la acción.

Paquete estimavp.dialog: es el paquete contiene los diálogos que se desea mostrar, los diálogos se muestra mediante el método performAction asociado a la clase de acción correspondiente al dialogo como es el caso de EstimavpFrConsultarEstimacion.java y EstimavpFrRealizarEstimacion.java.

Paquete estimavp.controlador: es el paquete que agrupa las clases EstimavpControlador.java, EstimavpCalisfot.java y EstimavpExportarPDF.java.

Paquete estimavp.util: es el paquete que agrupa todas las utilidades donde van estar las clases EstimavpTipoFactor.java, EstimavpConstantes.java y EstimavpTipoActividad.java.

Paquete estimavp.componentes: es el paquete contiene todos los componentes del visual.

Paquete estimavp.modelo: es el paquete que contiene el modelo de datos.

Estimavp.java: su función está dirigida a la carga y descarga del *plugin* mediante la clase VPPluginInfo que provee el openapi.jar, capturando la información definida en el archivo **plugin.xml.** EstimaVP.java implementa la interfaz VPPlugin definida en el openapi.jar la misma implementa los método **loaded()** y **unloaded()** para su función.

Plugin.xml: esta clase permite definir por medio de un script xml la configuración del *plugin* para ser cargado por la clase Plugin.java mediante la implementación de la interfaz VPPlugin.

Plugin.properties: su función es definir propiedades, asociados a los eventos y acciones del *plugin*.

EstimavpConsultarAccion.java: es la clase referente al control de las acciones de contexto la cual implementa la interfaz VPActionController definida por el openapi.jar para el manejo de las acciones para consultar la estimación.

EstimavpEstimarAccion.java: es la clase referente al control de las acciones a nivel de herramientas, la misma implementa la interfaz VPActionController definida por el openapi.jar para el manejo de acciones para realizar la estimación.

EstimavpFrConsultarEstimacion.java: representa el formulario asociado a la consulta de estimación, mostrando los datos generales del proyecto, generándolo a PDF.

EstimavpFrRealizarEstimacion.java: representa el formulario asociado a la realización de estimación así como las actividades, la cantidad de roles y los factores asociados.

EstimavpControlador.java: es la clase controladora de las acciones relevantes para las funcionalidades principales del *plugin.*

EstimavpCalisoft.java": esta clase que va a contener todos los métodos que tiene la herramienta de CALISOFT.

EstimavpExportarPDF.java: es la clase que exporta el reporte de la estimación en formato PDF.

EstimavpTipoFactor.java: es una clase enumerativa que contiene todos los tipos de factores asociados.

EstimavpConstantes.java: es la clase que contiene todas las constantes que están definidas en el proyecto.

EstimavpTipoActividad.java: es una clase enumerativa donde se clasifican todas las actividades.

EstimavpFactorEditor.java: es la clase que permite brindarle la funcionalidad a la tabla que va mostrar los valores de los factores, cuando la acción del mouse sobre el combox se muestra, cada vez que se haga un cambio que se escoja una opción, es la que admite modificar ese valor en el modelo en el factor que se está mostrando.

EstimavpFactorRender.java: es la clase que muestra el componente de la lista en la tabla que contiene los factores asociados a la estimación y le especifica a la tabla de que manera el contenido va en el campo, en este caso es un combox para poder coger la opción.

EstimavpProyecto.java: esta clase contiene todos los datos del proyecto.

EstimavpActividad.java: esta clase modela una actividad donde se define el tipo de actividad, el tiempo en horas, en semanas, en meses y en años.

EstimavpFactor.java: es la clase que implementa los tipos de factores y modela un factor con sus características.

EstimavpModeloFactor.java: es la clase que le va brindar el contenido del modelo que va a tener la tabla de realizar la estimación**.**

EstimavpFactorAbstracto.java: es una clase que modela los factores que influyen en las métricas.

EstimavpFactorCliente.java": esta clase hereda de la EstimavpFactorAbstracto.java e implementa el método asociado al factor cliente.

EstimavpFactorAmbiente.java: esta clase hereda de la EstimavpFactorAbstracto.java e implementa el método asociado al factor ambiente.

EstimavpFactorValorAgregado.java: esta clase hereda de la EstimavpFactorAbstracto.java e implementa el método asociado al factor valor agregado.

EstimavpFactorComplejidadTecnica.java: esta clase hereda de la EstimavpFactorAbstracto.java e implementa el método asociado al factor complejidad técnica.

EstimavResultadosMetricas.java: en esta clase es donde se va estar todo el resultado de las métricas.

openapi.jar: es la librería que ofrece *"Visual Paradigm for UML"* para la integración con el NetBeans.

Itextpd-5.2.1.jar: es la librería que permite generar el reporte de la estimación en formato de PDF.

AbsoluteLayout.jar: es una librería que posee por defecto el NetBeans, para posicionar de manera fija los componentes.

3.3. Pruebas de software

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Las pruebas son un proceso que se enfoca sobre la lógica interna del software y las funciones externas, es un proceso de ejecución de un programa con la intención de descubrir un error. Tiene éxito si descubre un error no detectado hasta entonces. Además, esta etapa implica: [23]

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.

- **Nivel de prueba**

Para comprobar que el *plugin* funcione de forma correcta, se realizan **Pruebas de Desarrollador** pues es el primer nivel de prueba y estas son diseñadas e implementada por el equipo de desarrollo.

- **Tipo de prueba**

Dentro de la pruebas de funcionalidad se realizan las **Pruebas Funcionales** que ejecutan a cada una de las funcionalidades para verificar que se cumplan los requisitos establecidos, incluyendo la navegación,

entrada de datos, procesamiento y obtención de resultados, se enfoca en los requisitos funcionales y casos de uso. El método que utiliza este tipo de prueba es el método de Caja Negra

- **Método de prueba Caja Negra**

El método de prueba a utilizar es el método de **caja negra** que consiste en las pruebas que se llevan a cabo sobre la interfaz del software. También conocidas como pruebas de comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. Las pruebas de caja negra pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto.

Para diseñar los casos de prueba de caja negra se utilizó la **Técnica de la Partición de Equivalencia** que divide el campo de entrada de un programa en variables de equivalencia con juegos de datos de entrada y salida. Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Se definen dos tipos de variables de equivalencia, las válidas, que representan entradas válidas al programa, y las no válidas, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor real de dato.

3.4. Casos de pruebas

A continuación se presentan los casos de pruebas, para los casos de uso Realizar Estimaciones y Consultar Estimación. El caso de uso Calcular Factores de Complejidad se realiza a través de la interfaz Realizar estimación por el Método Estimación UCI y es un incluido de Realizar Estimación. Estas pruebas se realizan a través de una matriz de datos, para comprobar que el *plugin* funcione de forma correcta, donde:

- V: indica válido, I: indica inválido, NA: que no es necesario proporcionar un valor del dato, ya que es irrelevante.
- En la siguiente tabla se muestran las variables que representan valores de entrada de datos para el caso de prueba Realizar Estimaciones y para el caso de uso Consultar Estimación no hay descripciones de variables porque no es necesarios entrar variables para probar la funcionalidad.

Descripción de las variables

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	profesional	campo de texto	no	Alfanumérico de 1 a 255 caracteres.
2	técnico	campo de texto	no	Alfanumérico de 1 a 255.
3	consultor	campo de texto	no	Alfanumérico de 1 a 255 caracteres.
4	pruebas de aceptación	campo de texto	no	Alfanumérico de 1 a 255 caracteres.
5	prueba piloto	campo de texto	no	Alfanumérico de 1 a 255 caracteres.
6	despliegue	campo de texto	no	Alfanumérico de 1 a 255 caracteres.
7	soporte	campo de texto	no	Alfanumérico de 1 a 255 caracteres.
8	factor ambiente	Combo Box	no	Selección (Si o No)
8	factor cliente	Combo Box	no	Selección (Si o No)
10	factor de complejidad técnica	Combo Box	no	Selección (A,MA,M,MB,B)
11	factor de valor agregado	Combo Box	no	Selección (Si o No)

Tabla 16: Descripción de las variables- Realizar Estimación.

Caso de prueba para el CU- Realizar Estimación

Descripción General:

El CU Realizar Estimación se inicia cuando el usuario selecciona trabajar en el *plugin* dentro del *"Visual Paradigm for UML"* para realizar la estimación.

Condiciones de ejecución

Que exista un diagrama de casos de usos con las especificaciones realizadas en el perfil diseñado. [Ver Anexo 1]

Caso de prueba para el CU- Consultar Estimaciones

Descripción General

El CU Consultar Estimación se inicia cuando el usuario selecciona la opción Consultar Estimación donde se consulta la estimación de algún proyecto después de haberse realizado la misma. En esta se interfaz genera el reporte de la estimación en formato PDF.

Condiciones de ejecución

Que se haya realizado la estimación del algún proyecto.

SC 1 Consultar Estimaciones

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Consultar Estimación correctamente.	Se consulta la estimación correctamente.	Se consulta correctamente la estimación del proyecto según los parámetros definidos en el Método de Estimación UCI.	1-Se abre el Visual Paradigm y se escoge la opción Consultar Estimación en la barra de la herramienta.
EC 1.2 Generar PDF. Consultar Estimación.	Se genera pdf correctamente.	Los datos de la estimación se han generado correctamente.	1-Se abre el Visual Paradigm y se escoge la opción Consultar Estimación en la barra de la herramienta. 2- Se muestra un botón para Generar PDF y otro para Cerrar.
EC 1.3 Cerrar. Consultar Estimación	Se cierra la operación consultar estimación.	Se cierra la operación de consultar estimación.	1-Se abre el Visual Paradigm y se escoge la opción Consultar Estimación en la barra de la herramienta. 2- Se muestra un botón para Generar PDF y otro para Cerrar.

Tabla 17: Matriz de datos para el CU- Consultar Estimaciones.

3.5. Resultado de las pruebas

Como resultado se realizaron pruebas funcionales a la aplicación, utilizando el método de caja negra aplicando la técnica partición de equivalencia. Además se diseñaron los casos de pruebas basados en cada caso de uso. En el CU Realizar Estimaciones se realizaron tres iteraciones por escenarios, detectando una no conformidad en la primera iteración, en el tercer y en la segunda iteración fue resulta, logrando comprobar la integración satisfactoria al *"Visual Paradigm for UML".*

3.6. Conclusiones parciales

Durante la realización de este capítulo se realizó el flujo de trabajo de implementación, en el cual se describieron los elementos del diseño en términos de componentes, así como las conexiones entre ellos. Se realizaron las pruebas funcionales donde quedaron definidos los diseños de casos de pruebas para los

cuales se tuvieron en cuenta las entradas, las salidas, los resultados esperados y el tratamiento de errores en caso de anomalías.

Conclusiones generales

Con el propósito de darle cumplimiento al objetivo general y a la problemática planteada en el presente trabajo, se han llevado a cabo satisfactoriamente cada uno de los objetivos específicos se puede establecer como conclusiones las siguientes:

- La caracterización del Método de Estimación UCI permitió su incorporación como *plugin* en *"Visual Paradigm for UML".*
- La elaboración del perfil UML permitió reflejar los datos en los diagramas que pueden ser utilizados en la aplicación del Metodo de Estimacion UCI.
- El desarrollo de los mecanismos de análisis, de diseño y de implementación estableció un factor fundamental para realizar estimación garantizando la trazabilidad de los requisitos funcionales.
- Aplicadas las técnicas de validación a la propuesta de solución, se comprobó el buen funcionamiento del sistema de acuerdo a los requisitos planteados.

Recomendaciones

Una vez vencidos los objetivos de esta investigación, y teniendo en cuenta las experiencias obtenidas a lo largo de su desarrollo se recomienda a la institución:

- Incorporar el método de estimación Post Arquitectura UCI que ayuden a contrastar el Método de Estimación UCI.

Referencias

1. informatica.uv.es/iiguia/2000/IPI/material/tema5.pdf

2. Difruta las Matemáticas.http://www.disfrutalasmatematicas.com/definiciones/estimacion.html.

3. Planificación de Proyectoshttp://ldc.usb.ve/~teruel/ci3715/clases/planif.html

4. pisuerga.inf.ubu.es/.../estimacion-del-esfuerzo-basada-en-casos-de-us...

5. rcci.uci.cu/index.php/rcci/article/download/66/60

6. Antonio Cicchetti, o., Automating Co-evolution in Model-Driven Engineering.: Italia.

7. Blanco, Y.G., Perfil de UML para los proyectos de la línea Soluciones Integrales. 2011, Universidad de las Ciencias Informáticas.: La Habana, Cuba

8 .OpenUP Version. Ayuda de OpenUP Version 1.5.0.1.

9 .Java. Los sistemas de información: importancia, fundamentos, calidad y gestión estratégica de las tecnologías de la información. Ecured 29 de noviembre del 2011 [cited

10. visual-paradigm.http://www.visual-paradigm.com/product/vpsuite/.

11. Netbeans.org. http://netbeans.org/index_es.html.

12. Una Introducción a los Perfiles UMLDpto. de Lenguajes y Ciencias de la ComputaciónUniversidad de Málaga

13.Ortiz,KadirHector.[Enlínea]

http://www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm. [Online]

14. requirements. http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf. [Online]

15. Anna. C Grimán, María Pérez, Luis. E Mendoza. Estrategia de Prueba para Software OO que garantiza Requerimientos No Funcionales.Caracas,Venezuela.

16. [cited; Available from: http://www.upedu.org/process/artifact/ar_desmd.htm

17. http://msdn.microsoft.com/es-es/library/bb972240.aspx

18. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.Design Pattens-Elements of Reusable Object Oriented Software

19. Ingeniería, Facultad de Informática-Universidad Politécnica de Madrid-Unidad Doc de.Patrones del "Grang of Four ". España-Madrid : s.n.

20. Patrones de Asignación de Responsabilidades. [cited 28 de abril de 2012]; Available from: http://www.ecured.cu/index.php/Patrones_de_Asignaci%C3%B3n_de_Responsabilidades

21. El Proceso Unificado de Desarrollo. s.l. : Addison Wesley. 2000.

22. El Arte de Modelar UML. 2007.

Bibliografías

1. informatica.uv.es/iiguia/2000/IPI/material/tema5.pdf

2. Difruta las Matemáticas.http://www.disfrutalasmatematicas.com/definiciones/estimacion.html.

3. Planificación de Proyectoshttp://ldc.usb.ve/~teruel/ci3715/clases/planif.html

4. pisuerga.inf.ubu.es/.../estimacion-del-esfuerzo-basada-en-casos-de-uso.

5. rcci.uci.cu/index.php/rcci/article/download/66/60

6. Ingeniería de Software I

7. OpenUP Version. Ayuda de OpenUP Version 1.5.0.1.

8 .visual-paradigm.http://www.visual-paradigm.com/product/vpsuite/.

9. Netbeans.org. http://netbeans.org/index_es.html.

10. biblioteca.uci.cu. [En línea] 2009. [Citado el: 9 de mayo de 2010.] http://biblioteca.uci.cu/sbd/biuci/index.html.

11.books.google.com.cu. [En línea] [Citado el: 13 de febrero de 2010.] http://books.google.com.cu/books?id=iBJstvkwFrYC&pg=PA274&lpg=PA274&dq=Los+problemas+de+estimaci%C3%B3n+surgen+en+todas+la+%C3%A1reas&source=bl&ots=5MsDS2GFyY&sig=tT8GO3BKmdWpY6H1QVq6boG8

12. flprincich.blogspot.com. [En línea] 16 de agosto de 2009. [Citado el: 16 de febrero de 2010.] http://flprincich.blogspot.com/2009/08/guias-practicas-de-estimacion-de.html.

13. SIRAC es una marca registrada del Centro Nacional de Producción más Limpia y Tecnologías Ambientales - CNPMLTA-SIRAC es una marca registrada del Centro Nacional de Producción más Limpia y Tecnologías Ambientales - CNPMLTA-http://www.sirac.info/Hospitales/html/indicadores.asp

14 .Ortiz, Kadir Hector. [En línea] http://www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm. [Online].

15. L. Cuaderno, o., Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools.: Argentina.

16. Una Introducción a los Perfiles UMLDepto. de Lenguajes y Ciencias de la ComputaciónUniversidad de Málaga.

17. requirements. http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf. [Online]

18. http://msdn.microsoft.com/es-es/library/bb972240.aspx

19. Ingeniería, Facultad de Informática-Universidad Politécnica de Madrid-Unidad Doc de.Patrones del "Grang of Four ". España-Madrid: s.n.

20. El Proceso Unificado de Desarrollo. S.l.: Addison Wesley. 2000.

21. El Arte de Modelar UML. 2007.

22. TiposPruebasSoftware.

23.5414_ MANUAL DE USUARIO DE LA HERRAMIENTA DEL MÉTODO DE ESTIMACIÓN UCI .

24. UCI, 5406_Bases del Método de Estimación.

25. UCI, 5414_Bases del Método de Estimación Post Arquitectura.

26. Danelys Zamora Suri y Alejandro Morales Torres.Tesis: Herramienta para la Estimación de los proyectos de desarrollo de software.

27. Radamés Linares Columbié, M.P.H., Larisa Viciedo Tijera. 9 de febrero del 2012 [cited; Available from: http://bvs.sld.cu/revistas/aci/vol8_3_00/aci09300.htm

Glosarios de Términos

- **Estimación:** estimar es deducir, apreciar y dar valor “Es una suposición cercana al valor real, normalmente, por medio de algún cálculo o razonamiento”.
- **Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*):** es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).
- **OMG *(Object Management Group):*** es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.
- **UML:** *(Unified Modeling Language)* Lenguaje Unificado de Modelado. Es el lenguaje de modelado de sistema de software más conocido en la actualidad.
- **Requisitos funcionales:** los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir.
- **Requisitos no funcionales:** los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable.
- **Matriz de trazabilidad:** la matriz de trazabilidad es una técnica que relaciona los requisitos funcionales a los diferentes elementos del desarrollo.
- **Pruebas de software:** la prueba del software es un elemento crítico para la garantía de la calidad del software.