Universidad de las Ciencias Informáticas

FACULTAD 6

Título: Extensión de la herramienta CASE “Visual Paradigm for UML” para la estimación de proyectos.

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

Autor: Dayana Carbonell Peró.

Tutores: Ing. Adaily Hernández Carballé.
Ing. Lianet Cabrera González

La Habana, junio del 2013
“Año 55 de la Revolución”

"Lo importante es no dejar de hacerse preguntas..."

Albert Einstein

Declaración De Autoría

Declaro ser autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Dayana Carbonell Peró

_____

Firma del Autor

Ing. Adaily Hernández Carballé

_____

Firma del Tutor

Ing. Lianet Cabrera González

_____

Firma del Tutor

Datos de Contacto

AUTOR:

Dayana Carbonell Peró.

Universidad de las Ciencias Informáticas.

La Habana, Cuba.

E-mail: dcarbonell@estudiantes.uci.cu

TUTORAS:

Ing. Adaily Hernández Carballé

Universidad de las Ciencias Informáticas.

La Habana, Cuba.

E-mail: acarballe@uci.cu

Ing. Lianet Cabrera González

Universidad de las Ciencias Informáticas.

La Habana, Cuba.

E-mail: lcabrerag@uci.cu

AGRADECIMIENTOS

En días como hoy es meritorio reconocer a esas personas que han estado ahí, paso a paso en cada batalla ayudándome a conquistar mi sueño:

A mi mamá, Maira, que siempre ha estado a mi lado, apoyándome, cuidándome y aconsejándome. Gracias por no dejar nunca de creer que podía lograrlo.

A mi papá, Yoel, que a pesar de todo me ha dado mucho más de lo que cualquiera esperaría, gracias por mantenerte firme y ayudarme a estarlo yo, gracias por estar hoy aquí.

A mi abuelo, Rey, que aunque no se encuentra ya entre nosotros, es el mayor responsable de que hoy este aquí. Por confiar siempre y quererme tanto, Gracias.

A mi novio, Pablo, que ha sido mi sustento, mi apoyo, mi consuelo en estos años de universidad. Gracias por ser mi amigo, mi compañero, mi protector. Gracias por creer en mí.

A Sheyla, que en poco tiempo se convirtió en una persona muy importante en mi vida, con la que he podido contar sin lugar a dudas. Gracias por la paciencia, por el apoyo, por ser tú.

A mi familia, por estar siempre ahí, dándome ánimos para no dejarme caer, a pesar de los contratiempos de la vida. Gracias por demostrarme lo fuerte que puedo ser.

A mis amigos por lidiar con mi carácter y aun así quererme cerca. Gracias Adrián, Felix, ustedes son como mi familia.

A mis amigos que desde la distancian me dan fuerzas para lograr mis metas: Amigos como Dainelis, Ernesto, María Isabel, Leodermis, Evelin, Carlos, Ariel.

A Bárbaro, por quererme y mimarme siempre, hacerme saber que nunca estaré sola, que siempre puedo contar con él.

A los profesores que siempre me apoyaron y tuvieron fe en mí cuando más lo necesitaba.

A todos los que de una forma u otra contribuyeron a que hoy este aquí ante ustedes, Gracias.

DEDICATORIA

Este trabajo de diploma va dedicado a una de las personas más importantes en mi vida. Un hombre que siempre creyó y confió en mí incondicionalmente, y al que desafortunadamente la vida no le alcanzó para acompañarme en este momento tan especial. Para ti abuelo, va este trabajo, que es la muestra de que siempre debemos mantenernos firmes a nuestros sueños… tú creíste en mí y aquí estoy… gracias por la confianza…

RESUMEN

Una de las tareas más importantes en el desarrollo de un software consiste en la realización del proceso de estimación. Este proceso permite predecir el tiempo, el esfuerzo, el costo y los recursos necesarios para producir el software, así como la viabilidad de su realización. La presente investigación se enmarcó en la concepción e implementación de una extensión para la herramienta CASE “Visual Paradigm for UML”, con el objetivo de realizar la estimación de un proyecto, aplicando el Método de Estimación Post-Arquitectura UCI. Con este fin se investigó el método de estimación antes mencionado y otros que le han servido de apoyo. Se realizó un estudio acerca de las herramientas existentes para realizar funciones semejantes. Finalmente, se obtuvo la extensión “EstimaPostArq” que permite realizar la estimación de proyectos, partiendo del diagrama de requisitos modelado en la herramienta CASE “Visual Paradigm for UML”. Además facilita que se exporte el resultado de la estimación en un documento con formato .pdf.

PALABRAS CLAVES

Diagrama de requisitos, extensión, factores de complejidad, Método de Estimación Post-Arquitectura UCI, métricas, “Visual Paradigm for UML”.

Tabla de Contenidos

ÍNDICE DE TABLAS

ÍNDICE DE IMÁGENES

INTRODUCCIÓN

La gestión de proyectos es una tarea importante en el desarrollo de productos de software, pues está basada en la organización y administración de recursos, de manera tal que se pueda culminar el trabajo requerido en el proyecto dentro del cronograma pactado (1). Dentro de la gestión de proyectos está contenida la etapa de planificación de proyectos. Esta establece las actividades que se deben realizar, su prioridad e interrelación (2). Además, estima costos y plazos del proyecto, identificando tareas y asignando tiempo y recursos a dichas tareas.

Los proyectos de desarrollo de software realizan el proceso de estimación para controlar y administrar los recursos que se necesitan, así como los utilizados antes y durante el proyecto. Existen numerosas variables humanas, técnicas, del entorno y políticas, que intervienen en su proceso y que a su vez pueden afectar los resultados finales (3). Sin embargo, cuando es llevada a cabo de forma sistemática, se logran resultados con un grado aceptable de riesgo, convirtiéndose en un instrumento útil para la toma de decisiones. (4)

El proceso de estimación es importante no solo para predecir el valor de variables concretas dentro de un proyecto, sino para determinar su viabilidad. No sería factible el inicio de un proyecto que probablemente fracase debido a que no cuenta con el tiempo, el esfuerzo o los recursos necesarios para ser llevado a cabo. La estimación de tiempo y esfuerzo es útil para la asignación de recursos, pues apoya la evaluación del impacto de los cambios y la reprogramación de un proyecto. Además facilita su gestión, apoya planificaciones realistas y objetivas, permitiendo resultados acordes con lo planificado (3).

La estimación es un proceso continuo, que acompaña todo el desarrollo del proyecto y comienza usando pocas variables en un nivel alto de abstracción. A esta primera etapa dentro de la estimación se la denomina macro estimación, la cual permite obtener valores aproximados de tiempo y esfuerzo para estudiar la viabilidad del proyecto. Una vez que este ha comenzado y se han obtenido los valores, se pueden efectuar comparaciones, detectar desvíos en el plan y realizar los ajustes correspondientes. A medida que el mismo progresa, aumenta su información, y los parámetros descriptivos de las etapas iniciales se convierten en otros más detallados y precisos (ejemplo la cantidad de módulos o el número de líneas de código) (5).

Con el objetivo de lograr productos de software de alta competencia y aceptación en el mercado, el estado cubano crea la Universidad de las Ciencias Informáticas (UCI), como parte de los programas de la Batalla de Ideas. Esta se alzó como un novedoso centro universitario para la formación de profesionales

comprometidos con su patria y altamente calificados en la rama de la Informática. Su misión es servir de soporte a la industria cubana de la informática, así como producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación. Para lograrlo se establece una estructura organizativa basada en centros de producción de software que vinculan la formación con la producción y la investigación. Como ejemplo se puede mencionar al Centro de Tecnología de Gestión de Datos (DATEC); que se encarga de investigar, desarrollar productos y brindar servicios relacionados con las bases de datos y análisis de datos. Está distribuido por departamentos entre los que se encuentra el Departamento Integración de Soluciones al cual tributa esta investigación.

Para lograr que los productos de software desarrollados en los diferentes departamentos de los centros productivos de la UCI tengan mejoras competitivas, se creó el "Centro Nacional de Calidad de Software", más conocido como CALISOFT. Su misión es ser líder en la gestión del conocimiento en el área de aseguramiento de la calidad del software (6). Dicho centro ha propuesto métodos para realizar el proceso de estimación de tiempo, costo, esfuerzo y tamaño para los productos de software. En estos momentos cuenta con una herramienta que lo realiza sustentándose en el Método de Estimación UCI, que es aplicado en las etapas iniciales del proyecto.

CALISOFT cuenta además con un segundo método de estimación, el Método de Estimación Post-Arquitectura UCI. Este es más preciso, pues parte de los artefactos definidos en la etapa de requisitos, concretamente del documento de Especificación de Requisitos de Software. Es aplicado en etapas más avanzadas en el desarrollo del software y permite contrastar la estimación inicial realizada con el Método de Estimación UCI. Actualmente en la institución, no se cuenta con herramientas web o de escritorio (desktop) que realicen este proceso. La información necesaria para realizar la estimación aplicando el Método de Estimación Post-Arquitectura UCI es recogida en un modelo (Excel), tornándose la tarea muy engorrosa. Por otra parte, el documento de Especificación de Requisitos de Software puede ser modificado a lo largo del desarrollo del proyecto, lo que constituye una posible modificación en otros artefactos que lo tienen a él como entrada. Teniendo en cuenta que para realizar la estimación aplicando el Método de Estimación Post-Arquitectura UCI, se utiliza la información recogida en el artefacto, se debe garantizar la actualización de este proceso sistemáticamente. Esto requiere un esfuerzo adicional, pues se deben digitar los datos en el modelo (Excel) constantemente.

En función de lo antes expuesto se identifica como **Problema de la investigación:** ¿Cómo agilizar el proceso de estimación de proyectos, partiendo de los requisitos del software, en el Departamento Integración de Soluciones de DATEC?

Para ello se determinó como **Objeto de estudio:** El proceso de estimación de proyectos de software; delimitado en el **Campo de acción:** Las herramientas de estimación de proyectos.
El **Objetivo general** de la investigación es: Desarrollar una extensión de la herramienta CASE "Visual Paradigm for UML" para la estimación de proyectos aplicando el Método de Estimación Post-Arquitectura UCI.

Para dar solución al objetivo general se definen los **Objetivos específicos:**

- Construir el marco teórico de la investigación para el desarrollo de la extensión.
- Realizar el análisis y diseño de la extensión de la herramienta CASE "Visual Paradigm for UML".
- Realizar la implementación y las pruebas funcionales de la extensión de la herramienta CASE "Visual Paradigm for UML".

Para cumplir los objetivos propuestos se identifican las siguientes **Tareas de investigación**:

- Selección de las herramientas y tecnologías a utilizar en la construcción de la solución.
- Identificación de los requisitos funcionales y no funcionales de la extensión.
- Realización del diagrama de casos de usos del sistema.
- Modelación de las clases del diseño.
- Descripción de la arquitectura de la extensión.
- Implementación de las métricas y factores de complejidad que utiliza el método de estimación Post-Arquitectura UCI.
- Definición de la estrategia de prueba a utilizar para comprobar el correcto funcionamiento de la extensión.
- Diseño de los casos de pruebas a aplicar.

La presente investigación está estructurada en tres capítulos:

- **Capítulo 1: Fundamentación Teórica**

En este capítulo se realizará un estudio del estado del arte sobre los temas relacionados con la planificación y estimación de proyectos de software, así como los conceptos básicos sobre los métodos de estimación existentes. Se definirán las tecnologías, técnicas y metodologías que brindarán soporte a la solución propuesta. Además, se incluirá el estudio del lenguaje de modelado UML y de la herramienta CASE "Visual Paradigm for UML" sobre la que se realizará la extensión.

- **Capítulo 2: Análisis y Diseño de la Extensión**

En este capítulo se analizarán y describirán las funcionalidades que debe tener la extensión. Se definirán los requisitos funcionales y no funcionales que se deben tener en cuenta para la implementación de la extensión. Serán especificadas las dependencias y relaciones con otro software a través de un diagrama conceptual del dominio de la extensión. Se identificarán actores, casos de uso y sus relaciones que definen la arquitectura de la misma. Será definido el diseño de clases con el propósito de representar la implementación del sistema y se describirán las clases más relevantes. Se elaborarán los diagramas de interacción, específicamente los diagramas de secuencia y se analizan los patrones de diseño y estilos arquitectónicos presentes en la implementación.

- **Capítulo 3: Implementación y Pruebas de la Extensión**

En este capítulo se presentará el modelo de implementación a través del diagrama de componentes. Será implementada la extensión aplicando para ello estándares de programación. Se comprobará el funcionamiento de la extensión y el cumplimiento de los requisitos de software anteriormente definidos, mediante la aplicación de pruebas funcionales a través del método Caja Negra.

CAPITULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

Las herramientas para la estimación de proyectos se han desarrollado progresivamente con el propio avance de la producción de software, garantizando un alto nivel de competitividad y mercado. Estas se han convertido en uno de los objetivos del Departamento de Integración de Soluciones de DATEC en nuestra universidad.

En el presente capítulo se realiza un estudio de los principales conceptos y aspectos significativos relacionados con el proceso de estimación de proyectos. En él se analizan las características y funcionalidades de algunos de los métodos de estimación conocidos, que permitirán formar las bases para una mayor comprensión del Método de Estimación Post-Arquitectura UCI. Se seleccionan las tecnologías, metodologías y herramientas a utilizar para el desarrollo del sistema.

1.1 El Proceso de Planificación como parte de la Gestión de Proyectos

La planificación se entiende como el proceso de plantearse un objetivo y proveerse de los medios para alcanzarlo. Se planifica con el propósito de reducir la incertidumbre sobre la base de un mejor conocimiento de la realidad y la previsión de lo que puede acontecer en caso de mantenerse algunas situaciones (2). La planificación comprende una serie de tareas entre las que se encuentran:

- Establecer el trabajo a realizar, restricciones y objetivos.
- Evaluar el tamaño de los productos de software y recursos.
- Producir el cronograma e identificar y fijar los riesgos.
- Iterar para obtener el plan de requisitos asignados.

La planificación implica una permanente adopción de posiciones. Es un conjunto de decisiones interrelacionadas y en progresión; una actividad continua y dinámica. Constituye un reajuste permanente entre medios, actividades, fines y procedimientos, dirigida hacia la realización de acciones que permiten el cumplimiento de los objetivos propuestos. Dentro de la gestión de los proyectos de software, la planificación es una tarea de vital importancia: "... es la compresión de planear un proyecto (plan de proyecto) y sus iteraciones (plan de iteraciones), lo que lleva a la planificación de las fases e iteraciones y cómo evaluar cada iteración" (7). La planificación es un área de proceso perteneciente a la categoría Gestión de Proyecto definida en el nivel de madurez dos del modelo de madurez CMMI[1] (2). Establece un

1 CMMI: *Capability Maturity Model Integration o* Integración de Modelos de Madurez de Capacidades.

marco de trabajo que permite al gestor de planificación hacer estimaciones razonables. De esta manera proporciona valores aproximados de recursos, costos y planificación temporal, que definan las actividades del proyecto.

1.2 El Proceso de Estimación de Proyectos de Software

Se considera imprescindible para planificar la aplicación de buenos métodos de estimación, por la creciente influencia que ejercen en el control preciso, predecible y repetido sobre los procesos de producción y los productos de software. De acuerdo con CMMI para servicios, estimación es el resultado probable calculado, que regularmente se aplica a cuestiones cuantitativas como los recursos de hardware y software, costos del proyecto, el esfuerzo y lapsos de tiempo. Es el cálculo de la duración, del esfuerzo y/o costo requerido para completar una tarea o un proyecto (8).

Estimar es anticipar con cierto grado de certeza los valores de los procesos, productos y recursos de un software y las características de éstos que implican mayor riesgo para el proyecto. Las estimaciones están basadas en probabilidades, debido a la influencia de factores externos de difícil control. Deben intentar definir los escenarios de mejor y peor caso de modo que los resultados del proyecto se puedan acotar (3).

1.3 Métodos de Estimación Proyectos de Software

En la actualidad existen diversos métodos de estimación de proyectos de software que toman en cuenta no solo los recursos, costos y planificación, sino también el procedimiento técnico a utilizar en el proyecto. Poseen objetivos o fines comunes, dentro de los que se encuentran: el establecimiento del ámbito del proyecto, así como la utilización de métricas de software y factores de complejidad. Se incluye además la descomposición del proyecto en pequeñas partes que permitan estimar tiempo, costos y recursos de manera individual. Entre estos métodos de estimación se encuentran:

1.3.1. Analogía

El método de estimación por Analogía constituye un complemento al de Juicio de Expertos; permite que las personas involucradas no sólo trabajen con su experiencia acumulada, sino que dispongan también de datos de proyectos ya finalizados equivalentes al que hay que estimar. Así, por comparación con proyectos similares en el mismo dominio de aplicación, se pueden evaluar las diferencias entre el nuevo proyecto y los antiguos, y extrapolar su costo (9).

La principal ventaja de la estimación por analogía es que está basada en la experiencia real de un proyecto, que puede ser estudiada para determinar las diferencias específicas con un proyecto nuevo y el

impacto de los cambios en los costos. Este método es bastante preciso si se disponen de datos de proyectos previos. Por otra parte, la principal desventaja es que no es evidente hasta qué punto es realmente representativo el proyecto previo, en lo que se refiere a restricciones, técnicas, personal y funcionalidad requerida (10).

1.3.2. Modelos Algorítmicos

Los modelos algorítmicos se apoyan en ciertos algoritmos matemáticos que realizan una estimación del costo del software en función de un número de variables consideradas como relevantes para el desarrollo de software. Los valores de los factores se establecen a partir del análisis de regresión de datos confiables recopilados en proyectos anteriores. Se le denominan drivers de costo y de su valor depende en gran medida la calidad del producto a desarrollar, así como el esfuerzo y el tiempo necesario para realizarlo (9).

Una de sus ventajas lo constituye la posibilidad de calcular el costo de las diferentes opciones que, aunque no sean exactas, pueden ser comparadas sobre una base objetiva; y permiten analizarlas cuantitativamente. La principal desventaja en el modelado algorítmico es que se basa en los atributos de los productos terminados para hacer la estimación. En las primeras etapas del proyecto, estos atributos son muy difíciles de estimar de forma infalible, lo que hace más difícil obtener datos precisos de la estimación realizada. Además, están calibrados a partir de experiencias anteriores. No se puede asegurar que estas experiencias sean realmente representativas de proyectos futuros; en especial si se desarrollan en nuevas áreas de aplicación, con nuevas técnicas y arquitecturas (9). El modelo COCOMO es un ejemplo de modelo algorítmico.

1.3.3. COCOMO II

COCOMO[2] es un modelo empírico que se obtuvo recopilando datos de varios proyectos grandes. Estos datos fueron analizados para descubrir las fórmulas que mejor se ajustaban. Dichas fórmulas vinculan el tamaño del sistema y del producto, factores del proyecto y del equipo, con el esfuerzo necesario para desarrollar el sistema (11).

Existe una segunda versión de este método, conocida por COCOMO II. La misma brinda la posibilidad de estimar el esfuerzo, costo y tiempo que se requiere en un proyecto de software, a partir de una medida del

[2] COCOMO: Constructive Cost Model o Modelo Constructivo de Costes

tamaño del mismo. Esta medida está expresada en el número de líneas de código que se estimen generar para la creación del software (12). De esta manera el método COCOMO II considera diferentes enfoques para el desarrollo de software. Engloba varios niveles que producen estimaciones detalladas de forma incremental; además soporta el modelo de desarrollo en espiral (11). Es un modelo bien desarrollado que tiene en cuenta el proyecto, el producto, el hardware y los atributos del personal.

COCOMO II ofrece tres etapas de modelos para la estimación de proyectos de software:

- **Composición del Modelo de Aplicación:** Se utiliza para las primeras fases o ciclos en espiral: prototipo de interfaz de usuario, así como cualquier otro prototipo que se produzca más tarde en el ciclo de vida del proyecto (12).
- **Modelo Inicial del Diseño**: Se aplica para las siguientes fases o ciclos en espiral. Involucra a la exploración de alternativas arquitectónicas o estrategias de desarrollo incremental. El nivel de detalle es coherente con el nivel de información disponible y el nivel general de estimación de la precisión necesaria en esta etapa. Permite obtener estimaciones aproximadas antes de que esté determinada por completo su arquitectura, basándose en puntos de función sin ajustar o líneas de código fuente (12).
- **Modelo Post-Arquitectura:** Una vez que el proyecto está listo para desarrollar y mantener un sistema alineado debe tener una arquitectura de ciclo de vida. Esto proporciona una información más precisa sobre los costos reales del proyecto (12).

Las ventajas de la aplicación de COCOMO II se ven reflejadas en su amplia aplicación en la industria del software. Es fácil de realizar y de interpretar; tiene pocas variables, además se acerca a la realidad en la mayoría de los casos (13). Dicho método correctamente calibrado puede ser muy preciso en las estimaciones. Sin embargo, el método es difícil de aplicar, los valores estadísticos requeridos son difíciles de calcular, sobre todo si no se cuenta con una base histórica de proyectos. A esta desventaja se suma la existencia de varios productos comerciales que lo implementan con buen soporte pero son productos caros (14).

1.3.4. Estimación por Puntos de Casos de Uso

La estimación por puntos de casos de uso es un modo de estimación temprana de cálculo del esfuerzo de desarrollo previsto para la ejecución de un proyecto de software. Sus principios derivan en cierto modo de la técnica de puntos de función. Esta estimación clasifica los casos de uso del sistema de software como:

simples, medios y complejos, y les asigna factores de peso de cinco, diez y quince respectivamente. Para determinar a qué clase se asigna cada caso de uso se consideran las transacciones que contiene; se toma como una transacción a cada evento identificado en el diagrama de casos de uso que ocurre entre el actor y el sistema. El método contempla la intervención de factores de complejidad técnica como: requisitos de seguridad, concurrencia, usabilidad y portabilidad. Examina además los factores ambientales del proyecto, entre los que se encuentran: estabilidad de los requisitos, motivación del equipo y experiencia con las herramientas (10).

Dentro de sus principales ventajas se encuentra que este método trabaja bien con diferentes tipos de software; muestra buen rendimiento en proyectos pequeños, medianos y grandes. Además se adapta rápidamente a empresas que ya estén utilizando la técnica de Casos de Uso (15). Entre sus desventajas podemos mencionar que no existe un estándar para escribir casos de uso lo que dificulta la aplicación del método. Sólo se aplica al esfuerzo requerido para la fase de codificación del proyecto, siendo necesario aplicar otros ajustes con el fin de obtener el esfuerzo de todo el ciclo de vida del proyecto. Además las herramientas en esta área son caras y se enfocan en la evaluación del proyecto (16).

1.3.5. Métodos de estimación propuestos por CALISOFT

En la UCI existía la necesidad de contar con métodos de estimación que se ajustaran a las características de los productos de software que se desarrollan en sus centros productivos. Teniendo en cuanta esto CALISOFT crea un método de estimación, basándose en los métodos explicados anteriormente. Dicho método adopta el nombre de Método de Estimación UCI, que se describe a continuación:

- **Método de Estimación UCI**

El desarrollo del Método de Estimación UCI parte de la necesidad en la universidad de estimar el tiempo y esfuerzo requeridos para desarrollar un producto de software. En este método se tienen en cuenta los tiempos a emplear y el esfuerzo de cada rol, en dependencia de la complejidad de los casos de uso. La complejidad de los casos de uso se determina de una manera empírica e intuitiva, pues en la UCI no se contaba con una vasta experiencia en las estimaciones de proyectos (10). Este método cuenta con atributos de estimación como: tamaño, tiempo, esfuerzo, recursos y costo en ese orden, debido a que la estimación de uno depende del anterior. Se tienen 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. En dicho método se identifican las etapas por las que debe pasar el proceso de

desarrollo de software y las actividades incluidas en cada una de ellas. Del mismo modo, la cantidad de horas empleadas en dependencia de la complejidad Baja, Media o Alta por casos de uso identificados. Se tiene en cuenta además los roles que intervienen según la etapa a medir (10).
Este método ofrece ventajas para el desarrollo de productos de software, pues permite realizar una estimación temprana, con la cual se obtienen elementos que permitan determinar los costos y recursos aproximados para la realización de los mismos.

Teniendo en cuenta que el Método de Estimación UCI se aplica en etapas tempranas del desarrollo de software cuando aún no se ha definido la arquitectura del sistema, el resultado de la estimación es poco preciso. Debido a esto CALISOFT propone un segundo método de estimación, que tiene como punto de partida los resultados del Método de Estimación UCI. Este método recibió el nombre Método de Estimación Post-Arquitectura UCI.

- **Método de Estimación Post-Arquitectura UCI**

El Método de Estimación Post-Arquitectura UCI, es el método que se selecciona para el desarrollo de la extensión. Se aplica en etapas más avanzadas en el desarrollo del software y permite contrastar la estimación inicial realizada con el Método Estimación UCI. El mismo parte de la necesidad de contar con un segundo método de estimación en la UCI, que sea más preciso, a partir de la definición de los requisitos del software que se va a desarrollar. Se caracteriza por poseer factores de complejidad como: Factor de Complejidad Técnica y el Factor de Valor Agregado. Estos factores influyen en las métricas de tiempo, costo, esfuerzo, tamaño y recursos necesarias para el cálculo de la estimación. Dado que la universidad no cuenta con una base histórica referente a la estimación de proyectos, se tomaron como punto de partida los resultados arrojados por el TWG (Technical Working Group o Grupo de Trabajo Técnico) de Requisitos; además de los resultados del Método de Estimación UC (17)I.
Para la aplicación de este método se deben tener en cuenta factores de complejidad y métricas que son imprescindibles para realizar el proceso de estimación.

Factores de complejidad que influyen en las métricas

Los factores de complejidad definidos para llevar a cabo la segunda estimación se listan asignando un peso determinado en una escala de cero coma cinco a dos 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 (17):

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: Peso de los factores de complejidad

Escala de 0-1:

Evaluación	Valoración
Si	1
No	0

Tabla 2: Valoración cualitativa de cada factor

Luego de clasificar la complejidad por cada factor se procede a calcular el resultado final para cada uno; para ello se multiplica el peso predeterminado para cada factor por 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 (17).

Para este método solo se tuvieron en cuenta los factores de valor agregado y complejidad técnica, ya que a partir del análisis realizado, solo estos dos siguen influyendo en las estimaciones.

Factor de Valor Agregado (VA)

Mediante el Factor de Valor Agregado se determinan 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 casos donde el nivel de desarrollo sea alto (17).

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

Tabla 3: Factor de Valor Agregado

P*V: Peso * Valoración.

El Factor Valor Agregado se calcula como: uno más la sumatoria del producto de cada factor individual por su respectivo peso **(1+ ((AI*P) + (IFN*P) + (AM*P)))** convertido a una escala entre uno y uno coma cinco mediante la fórmula: **VA = 1+FO*0.2/FP**, Donde:

VA: Valor Agregado

FO: Factor obtenido: **((AI*P) + (IFN*P) + (AM*P))**.

FP: Factor obtenido asignando valoraciones pesimistas (17).

Factor de Complejidad Técnica (CT)

Es asociado a la complejidad de la tecnología y requisitos no funcionales del sistema. Influye directamente en el esfuerzo y fundamentalmente en la etapa de implementación (17).

No.	Factor	Descripción	Peso	Complejidad	P*V
CT 1	Rendimiento del sistema (RS).	Velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia del sistema.	1	(A, MA, M, MB, B)	
CT 2	Rendimiento de la plataforma (RP).	Velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia de la plataforma.	1	(A, MA, M, MB, B)	
CT 3	Complejidad de diseño gráfico (CDG).	Nivel de complejidad en el conjunto de operaciones 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 (PIC).	Procesamientos lógicos o matemáticos intensivos en el sistema.	1	(A, MA, M, MB, B)	

CT 5	Incluye objetivos especiales de seguridad (OES).	La disponibilidad de mecanismos que controlen o protejan los programas o los datos.	1	(A, MA, M, MB, B)	
CT 6	Reutilización (R).	Definida por una métrica específica.	0.5	Según fórmula	
CT 7	Integración (I).	Definida por una métrica específica.	1	Según fórmula	
CT 8	Asimilación de sistemas (AS).	Definida por una métrica específica.	1	Según fórmula	
CT 9	Asimilación de dispositivos (AD).	Definida por una métrica específica.	0.5	Según fórmula	
CT10	Asimilación de estándares (AE).	Definida por una métrica específica.	1	Según fórmula	

Tabla 4: 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 lo cual existe una escala para llevarlas a la ecuación cualitativa (17):

- **Porciento de Reutilización** = CMR/CMD
  CMR: Cantidad de requisitos con reutilización.
  CMD: Cantidad de requisitos a desarrollar.
- **Porciento de Integración** = CSI / CMS
  CSI: Cantidad de sistemas a integrar o legar.
  CMS: Cantidad de requisitos a desarrollar.
- **Porciento de Asimilación de Sistemas** = CSI-D / CSI
  CSI: Cantidad de sistemas a integrar o legar.
  CSI-D: Cantidad de sistemas a integrar o legar desconocidos.
- **Porciento de Asimilación de Dispositivos** = (CDE – DRV) / CDE
  CDE: Cantidad de dispositivos externos.
  DRV: Cantidad de dispositivos externos con driver o SDK.
- **Porciento de Asimilación de Estándares** = CEE / CEC
  CEC: Cantidad de estándares y normas a cumplir.
  CEE: Cantidad de estándares y normas específicas.

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. Finalmente, el valor de la Complejidad Técnica se calcula como: uno más la sumatoria del producto de cada factor individual por su respectivo peso convertido a una escala entre uno y dos mediante la fórmula: $CT = 1+FO*0.5/FP$, Donde:
CT: Complejidad técnica.
FO: Factor obtenido: **((RS*P)+(RP*P)+(CDG*P)+(PIC*P)+(OES*P)+(R*P)+(I*P)+(AS*P)+ (AD*P)+(AE*P)).**
FP: Factor obtenido asignando valoraciones pesimistas (17).

El Método de Estimación Post-Arquitectura UCI define además un conjunto de métricas que deben ser calculadas para obtener el resultado final de la estimación.

Métricas definidas en la segunda estimación

El primer paso del método es la identificación de la cantidad de requisitos que potencialmente tendrá el sistema a desarrollar. Estos se deben definir previamente en el documento Especificación de Requisitos, durante la etapa Captura de Requisitos. Una vez definidos, se evalúan según su complejidad. El peso de los requisitos se determina utilizando los elementos especificados en la siguiente tabla (17).

Peso máximos y mínimos de los requisitos según su complejidad		
Clasificación	**Cota Mínima**	**Cota Máxima**
Bajo	1	1.4
Medio	1.5	2.4
Alto	2.5	3

Tabla 5: Peso máximo y mínimo de los requisitos según su complejidad

Las métricas utilizadas para realizar el proceso de estimación aplicando este método son:

Métricas de Tamaño

Para unificar los requisitos según la complejidad, se aplicará una métrica donde influye el peso definido, quedando (17):

TRA= TotRA *CMaxA, Donde:
TRA: Tamaño de requisitos de complejidad Alta.

TotRA: Total de requisitos de complejidad Alta.
CMaxA: Peso Máximo de requisitos de complejidad Alta.

TRM= TotRM *CMaxM *FC, Donde:
TRM: Tamaño de requisitos de complejidad Media.
TotRM: Total de requisitos de complejidad Media.
CMaxM: Peso Máximo de requisitos de complejidad Media.

TRB = TotRB *CMaxB, Donde:
TRB: Tamaño de requisitos de complejidad Baja.
TotRB: Total de requisitos de complejidad Baja.
CMaxB: Peso Máximo de requisitos de complejidad Baja.

Obtenido el valor del tamaño para cada tipo de requisito se procede a calcular el tamaño total (17):
UDA= Σ (TRA + TRM + TRB)*CT, Donde:
UDA: Total de Unidades de Desarrollo Ajustado.

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 (17).

Porciento de tiempo por actividad respecto al total de tiempo de desarrollo	
Análisis y diseño	17.12%
Implementación	31.32%
Pruebas Internas	10.10%
Pruebas de Liberación	3.08%

Tabla 6: Porciento de tiempo por actividad respecto al total de tiempo de desarrollo

Dado estos ajustes, y teniendo en cuenta que el proyecto ha pasado ya por las tres primeras etapas, las estimaciones estarán en función del 61.62 % (17).

Para calcular Tiempo Total de Desarrollo se inicia por el cálculo del tiempo de la etapa de implementación (TImp), teniendo en cuenta que el tiempo más fácil de estimar es el de implementación de un requisito.

TImp = TImpUnitario * UDA Donde:

TImpUnitario: Constante que identifica el tiempo de implementación de un requisito.

TImpUnitario=35 horas a tiempo completo y 42 horas a medio tiempo dedicado a la implementación (17).

Después se calcula el tiempo de Análisis y Diseño (TAD) utilizando los porcientos definidos para cada etapa en la tabla, agregándole el factor de valor agregado (FAG).

TAD = (TImp * %AD)/%Imp)* FAG, Donde:

%AD: Constituye el porciento de tiempo correspondiente a la etapa 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).

El tiempo dedicado a las Pruebas Internas se determina mediante la fórmula:

(TPI)= (TImp * % Pruebas Internas)/ % TImp.

El tiempo dedicado a las Pruebas de Liberación puede calcularse por:

(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) (17).

TRD = E/TH

Métrica del Esfuerzo

Para calcular el Esfuerzo de Desarrollo (E) se tendrán en cuenta todos los tiempos dedicados a las etapas.

E= Σ(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. El peor de los casos corresponde a cuando hay involucrados a tiempo medio, en el cual el promedio es de cinco horas. Por otro lado, el mejor de los casos es cuando la vinculación a la producción es a tiempo completo, el promedio es de ocho horas. También se tendrá en cuenta la cantidad de días laborables a la semana que son cinco coma cinco y el tiempo límite de solución del proyecto (17).

EMax= HD*DL*TLS, Donde:

HD: Tiempo de trabajo diario.

DL: Cantidad de días laborables a la semana.

TLS: Tiempo Límite de Solución.

Nota: La UCI ha estipulado que los proyectos no deben excederse de un año, siendo este el valor de TLS.

Para calcular el Tiempo Máximo de Realización del Proyecto se tendrán en cuenta:

TMax= TRD + TTE*8*5, Donde:

TMax: Tiempo Máximo.

TTE: Es la sumatoria del tiempo pactado con el cliente para las actividades de Pruebas de Aceptación, Piloto, Despliegue y Soporte.

El Tiempo Máximo representa la estimación más certera de lo que se puede tardar el proyecto, representado en horas, semanas y meses (17).

Métricas de Recursos

El Total de Hombres (TH) necesarios para el desarrollo del software se obtendrá a partir del esfuerzo de desarrollo y el esfuerzo máximo de desarrollo del tiempo real de desarrollo (TRD (17)).

TH= E/EMax

Métricas de Costo

Para calcular el Costo de Desarrollo de Software (CS) se necesita partir del tiempo que llevará el desarrollo del proyecto, las tarifas asociadas a la cantidad de roles estimados inicialmente por el Jefe de Proyecto, y el Total de Hombres necesarios (representan el rol de los Profesionales). Los roles identificados serán clasificados en: Técnico, Profesional o Consultor de Alto Nivel a los que se les asocia una tarifa presupuestada con una cota mínima y otra máxima (17).

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 7: Tarifas por Rol

El costo del software se calcula a partir del TRD como:

CS= (TMax * TotalConsultores * TarifaConsultores) + (TMax * TotalTécnicos * TarifaTécnicos) + (TMax * TotalProfesionales * TarifaProfesionales)

Esta es la interpretación por pasos de la aplicación de este método a partir de la definición de los requisitos del software, que deja ver entre sus ventajas: mayor precisión y confiabilidad de sus resultados.

1.4 Lenguaje de modelado UML 2.0

UML (Lenguaje Unificado de Modelado o Unified Modeling Language) es un lenguaje de modelado de sistemas de software respaldado por el OMG (Object Management Group) y aprobado por éste como un estándar. Es un lenguaje gráfico elaborado para visualizar, especificar, construir y documentar un sistema de software. A través del modelado se reduce el ámbito del problema de estudio al enfocar solo un aspecto a la vez (18). Ofrece un estándar para describir modelos del sistema, incluyendo aspectos conceptuales tales como procesos de negocios, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. UML cuenta con una sintaxis y una semántica con reglas sobre cómo agrupar los elementos del lenguaje y el significado de esta agrupación (lenguaje). Es visual, mediante su sintaxis se modelan distintos aspectos de situaciones reales, que permiten una mejor interpretación y entendimiento (modelado). Además, unifica varias técnicas de modelado en una sola (unificado) (18).

Entre las principales ventajas que tiene la utilización de UML en la construcción de sistemas de software se encuentran:

- **Mejores tiempos totales de desarrollo (de 50% o más):** Con el uso de UML las fases de análisis y diseño consumirán mayor tiempo, pero el tiempo de construcción, implantación y estabilización se reducen drásticamente debido a que no hay correcciones mayores en las fases de mayor impacto de un proyecto.
- **Mejor calidad:** El uso de UML hace indispensable la participación del usuario en la definición de requisitos y por lo tanto mejora considerablemente el apego del sistema a las necesidades de sus usuarios. El mantenimiento correctivo se reduce drásticamente (hasta un 80% con respecto a un sistema hecho sin metodología). Algo similar ocurre en los proyectos de reingeniería.
- **Mejor soporte a la planeación y al control de proyectos:** Al existir entregables definidos y estandarizados en las distintas fases de un proyecto, y al ser estos revisables y certificables por

personas distintas del autor; los planes de trabajo pueden ser fácilmente creados y corroborados en avance, permitiendo tomar decisiones a tiempo.

- **Mayor independencia del personal de desarrollo:** Al tener documentadas las aplicaciones en un lenguaje estándar, es posible mover al personal de una aplicación a otra sin correr altos riesgos y sin depender del conocimiento personal de las aplicaciones.
- **Mayor soporte al cambio organizacional, comercial y tecnológico:** Un modelo permite cuantificar el impacto de un cambio antes de hacerlo y permite ensayar distintos enfoques de solución. Con UML un cambio se puede hacer primero en papel.
- **Alta reutilización:** Los productos de un desarrollo pueden ser usados en otro. Se pueden crear componentes reusables que con la difusión y administración adecuadas minimizarán costos y errores.
- **Minimización de costos:** Los puntos antes mencionados tienen un impacto económico que generalmente tiende a ser proporcional al tamaño de la organización.

UML se puede utilizar con cualquier metodología, ya sea el ciclo en cascada, en espiral, o incluso metodologías ágiles, porque es independiente del ciclo de desarrollo que se vaya a seguir. No especifica en sí mismo qué metodología o proceso usar. Permite hacer modelos que visualicen cómo es un sistema o cómo se quiere que sea y así especificar la estructura y/o comportamiento de este. UML cuenta con varios tipos de diagramas. Cada uno de ellos representa o modela un aspecto del sistema a través de las diferentes entidades representadas. Puede ser usado extensivamente en: recopilación de requisitos, análisis de aplicaciones, diseño de sistemas, en pruebas, en implementación, en reingeniería y prácticamente en cualquier actividad de desarrollo que sea susceptible de ser modelada (18).

1.4.1 Diagrama de requisitos

La herramienta CASE “Visual Paradigm for UML” proporciona un diagrama de requisitos SysML[3] para la especificación y análisis de requisitos (19). Este es un diagrama personalizado usado para describir los requisitos o características de un sistema como un modelo visual que no se reflejan en diagramas de casos de uso ni en otros diagramas UML tradicionales. Este diagrama permite la especificación y captura de requisitos de forma efectiva y práctica; además, permite mostrar de forma gráfica los requisitos en un alto nivel de abstracción. El diagrama de requisitos se compone por clases y sus relaciones (19). Estas

[3]SysML: *Systems Modeling Language* o Lenguaje de Especificación de Sistemas.

clases representan los requisitos del sistema que poseen atributos por defecto como: nombre, id, texto, tipo de requisito, método de verificación y riesgo; que constituyen descripciones del requisito representado. A estas clases se les pueden añadir nuevos atributos, como es el caso de la complejidad. Este atributo debe ser añadido por el usuario especificando su valor A, M, B que se corresponde con Alto, Medio o Bajo respectivamente. El mismo es la clave para realizar la estimación de proyecto empleando la extensión que aplica el método Post-Arquitectura UCI. A continuación se muestra una imagen que representa un ejemplo del diagrama de requisitos antes mencionado:

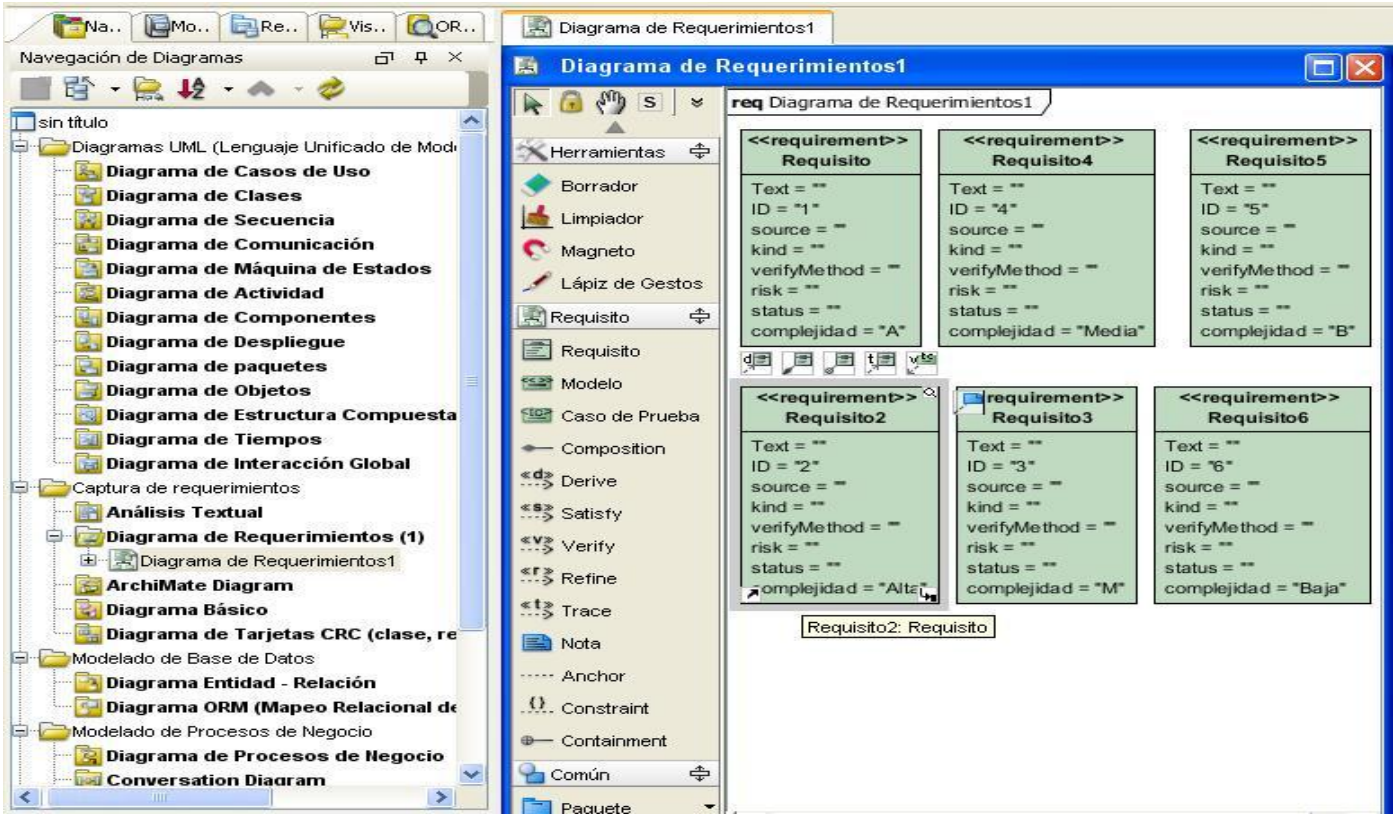


Imagen 1: Diagrama de Requisitos

1.5 Metodología para el desarrollo del Software

La metodología para el desarrollo del software Open Up (Open Unified Process o Proceso Unificado Abierto) se escoge para el desarrollo de la extensión, teniendo en cuenta las características que posee: Constituye un modelo extensible, dirigido a gestión y desarrollo de proyectos de software; equilibra las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto; además se centra en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo evolutivo, permitiendo

obtener retroalimentación y mejoramiento continuo de los procesos. Open UP estructura el ciclo de vida de un proyecto en cuatro fases: concepción, elaboración, construcción y transición. El ciclo de vida del proyecto provee a los interesados un mecanismo de supervisión y dirección para controlar los fundamentos del proyecto, su ámbito, la exposición a los riesgos y el aumento de valor (20).

Esta metodología ofrece enormes ventajas, pues se basa en el desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos. Es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo. Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas. Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP (Rational Unified Process o Proceso Unificado Racional) (21).

1.6 Lenguaje de Programación Java

Entre los lenguajes de programación orientado a objetos se encuentra Java, que ha sido seleccionado para la implementación de la extensión, pues es el lenguaje de programación que propone en API[4] de desarrollo de la herramienta CASE “Visual Paradigm for UML“. Java permite la modularidad, por lo que se pueden hacer rutinas individuales que sean usadas por más de una aplicación. Está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red y estaciones de trabajo. Además, se puede utilizar sobre arquitecturas distintas y con sistemas operativos diversos (22). Fue diseñado para crear un software altamente fiable, para lo cual proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros). Esto se debe a que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria. Para el desarrollo de aplicaciones utilizando el lenguaje de programación Java, se necesita la Máquina Virtual de Java (22).

La Máquina Virtual de Java 1.6.1

La Máquina Virtual Java (JVM del inglés Java Virtual Machine) permite que los proyectos sean ejecutados en diferentes entornos de hardware o software, además incluye en su concepción filosófica el colector de basura, facilitando que los implementadores se abstraigan del ciclo de vida de los componentes.Su misión principal es garantizar la portabilidad de estas aplicaciones; para lo que define esencialmente un

[4]Application Programming Interface o Interfaces de Programación de Aplicaciones (API): Conjunto de funciones que facilitan el intercambio de mensajes entre dos sistemas, es decir, que permiten la integración de diversos módulos a sus aplicaciones principales.

ordenador abstracto y especifica las instrucciones (bytecodes) que este ordenador puede ejecutar. Se vinculan las técnicas de compilación (intérprete, compilador) que se ejecutan sobre la JDK[5], además, permite la búsqueda y tratamiento de errores en el proceso de compilación. Entre las tareas principales de la JVM se encuentran (23):

- Reservar espacio en memoria para los objetos creados.
- Liberar la memoria no usada (garbage collection).
- Asignar variables a registros y pilas.
- Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos.
- Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java.

1.7 Herramientas a utilizar para el desarrollo de la Extensión.

Para el desarrollo de la extensión se emplearon herramientas que facilitaron su diseño e implementación. Entre ellas se encuentran la herramienta de modelado Visual Paradigm for UML y la plataforma de programación NetBeans IDE, que se definen a continuación:

- **Visual Paradigm for UML 8.0**

Para el desarrollo de la extensión se utilizó la herramienta “Visual Paradigm for UML”, pues permite de manera rápida, dinámica e interactiva generar los diagramas de requisitos, que serán utilizados para determinar la estimación mediante el Método Post-Arquitectura UCI. Se tuvo en consideración además que esta constituye la herramienta que utiliza la Universidad para modelar el desarrollo de software, especialmente el centro DATEC. “Visual Paradigm for UML” es una herramienta CASE muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Este software modela el UML para la construcción de aplicaciones, permitiendo dibujar diagramas UML. Fue creado para el ciclo vital completo del desarrollo del software, que lo automatiza y acelera, facilitando la captura de requisitos, análisis, diseño e implementación (19).

Visual Paradigm for UML también proporciona características tales como generación del código, ingeniería inversa y generación de informes. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. La herramienta está diseñada para usuarios interesados en sistemas de software de gran escala, con el uso del acercamiento

[5]Java Development Kit (JDK): Software que proporciona herramientas de desarrollo para la creación de programas en Java.

orientado a objetos; además, apoya los estándares más recientes de las notaciones de Java y de UML, incorporando el soporte para trabajo en equipo (24).

- **NetBeans IDE 7.2.1**

El NetBeans IDE[6] es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Es una aplicación de código abierto ("open source"), escrita en Java que soporta el desarrollo de aplicaciones en este y otros lenguajes (PHP, JavaScript, ExtJS). Es también un GUI (Graphical User Interface o Interfaz Gráfica de Usuario) diseñado para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología Java. Cuenta con un editor multicódigo, multiplataforma y autocompletado de código. Posee una interfaz gráfica muy completa y un escritorio de trabajo organizado, pues divide el entorno de desarrollo en marcos o ventanas que pueden ajustar su tamaño o incluso ocultarlas para que haya más espacio y claridad (25). NetBeans IDE dispone de soporte para crear interfaces gráficas de forma visual y sus funcionalidades son ampliables mediante la instalación de paquetes (packs) (26).

Por las características del Netbeans IDE 7.2.1 de ser multiplataforma y compilación del lenguaje Java, que es el lenguaje propuesto por el API de la herramienta CASE “Visual Paradigm for UML”, además de las mencionadas, se elige esta herramienta para desarrollar la extensión propuesta.

Conclusiones Parciales

En este capítulo se analizaron los conceptos relacionados con la gestión de procesos y la planificación, lo que permitió entender mejor el proceso de estimación de proyectos de software. Se realizó un estudio de los métodos de estimación utilizados como base por CALISOFT para la creación del Método de estimación Post-Arquitectura UCI, lo que facilitó el entendimiento del mismo. Se hizo referencia a las tecnologías y herramientas que se utilizarán para el desarrollo de la extensión; particularmente el diagrama de requisitos, que le permitirá a la extensión la obtención de datos imprescindibles para el cálculo de la estimación. Además se analizaron las características del proyecto, lo que permitió seleccionar la metodología de desarrollo de software más apropiada, así como el lenguaje de programación que mejor se corresponda.

[6]IDE: Entorno de Desarrollo Integrado

CAPITULO 2: ANÁLISIS Y DISEÑO DE LA EXTENSIÓN

Introducción

En el presente capítulo se realizará un análisis de las características que debe tener la extensión, definiendo para ello los requisitos funcionales y no funcionales. Serán definidos y descritos los casos de uso, identificando actores y su relación con éstos. También se especificarán las dependencias y relaciones con otro software. Se analizarán los patrones de diseño y estilos arquitectónicos. Además, serán diseñados los diagramas de clases que describen la implementación de la extensión.

2.1 Modelo de dominio

En el siguiente modelo de dominio se capturan los tipos más importantes de objetos en el contexto del sistema. Se representa como un diagrama de clases que contiene, no conceptos propios de un sistema, sino de la propia realidad física. Su objetivo es comprender y describir las clases más significativas del sistema y se toma como el punto de partida para el diseño de éste. Cuando se realiza la programación orientada a objetos, el funcionamiento interno del software va a imitar en alguna medida a la realidad. De esta manera el mapa de conceptos del modelo de dominio constituye una primera versión del sistema (27).

En este caso se refleja cómo el usuario, al utilizar la herramienta CASE “Visual Paradigm for UML”, podrá realizar el modelado de diagramas. Entre los diagramas que se modelan se encuentra el Diagrama de Requisitos. Este posee varios atributos entre los que está la complejidad. En la herramienta se podrá realizar la estimación de un proyecto mediante la utilización de la extensión EstimaPostArq, así como generar el reporte de ésta. La extensión utilizará los datos capturados del diagrama de requisitos que se modele en la herramienta y empleará el Método de Estimación Post-Arquitectura UCI. Este método de estimación está compuesto por métricas de costo, recursos, esfuerzo, tiempo y tamaño. Incluye además factores de complejidad que influyen en dichas métricas: el factor de complejidad técnica y el factor de valor agregado. Los datos capturados de los diagramas y la información recogida del método de estimación se procesan en la extensión, permitiendo que se muestre finalmente el reporte de la estimación realizada.

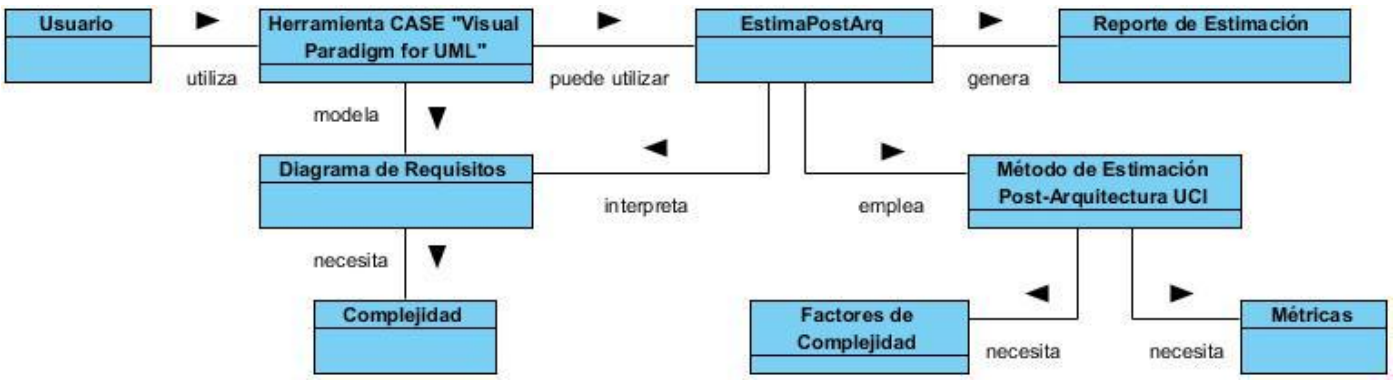


Imagen 2: Modelo de Dominio

2.1.1 Definición de las clases del modelo del dominio

Usuario: Persona que interactúa con la herramienta CASE "Visual Paradigm for UML" y la utiliza como herramienta de modelado para el análisis y diseño de proyectos de software.

Herramienta CASE "Visual Paradigm for UML": Herramienta CASE que permite el modelado de procesos del negocio a través de diagramas. Es la herramienta a la que se le realizará la extensión para el cálculo de la estimación de proyectos de software y en la que se modelan los diagramas que son utilizados para este proceso.

Diagrama de Requisitos: Representación gráfica utilizada para la especificación y análisis de los requisitos de un proyecto de software. Es creado en un contenedor de requisitos de la barra de herramientas lateral de la herramienta CASE "Visual Paradigm for UML" para representar la relación entre los requisitos y los demás elementos del modelo. Es el diagrama a través del cual se obtendrán las métricas necesarias para realizar el proceso de estimación.

Complejidad: Atributo de la clase representativa de un requisito en el diagrama de requisitos. Define la complejidad de este requisito según las clasificaciones de Alta, Media o Baja.

EstimaPostArq: Nombre de la extensión de la herramienta CASE "Visual Paradigm for UML" que captura los datos del diagrama de requisitos para realizar la estimación.

Reporte de Estimación: Representa el resultado generado mediante la realización de la estimación por la extensión y que es visualizada por el usuario.

Método de Estimación Post-Arquitectura UCI: Método de estimación desarrollado en la UCI por el centro CALISOFT que es utilizado en la extensión para realizar la estimación a partir de la definición de los requisitos del software a desarrollar y que son obtenidos del diagrama de requisitos. Para realizar el cálculo de la estimación mediante este método se calculan las métricas y los factores de complejidad que influyen en estas.

Factores de Complejidad: Constituyen factores que influyen en el cálculo de las métricas. Estos factores se listan, dándoles un peso determinado según una escala de cero coma cinco a dos en dependencia del nivel de importancia que tengan. El resultado final será la sumatoria del valor calculado para todos los criterios. Existen dos tipos de factores de complejidad: **Factor de Valor Agregado** que es el factor que determina acciones que pueden representar un esfuerzo adicional en el proceso de desarrollo de software. Está presente en todas las etapas de este proceso, especialmente en la de Análisis y Diseño. Además está el **Factor de Complejidad Técnica** que es asociado a la complejidad de la tecnología y requisitos no funcionales del sistema. Influye directamente en el esfuerzo y está presente fundamentalmente en la etapa de implementación.

Métricas: Parámetros que contribuyen al cálculo de la estimación. Luego de identificada la cantidad de requisitos que potencialmente tendrá el sistema y evaluados según su complejidad se determina un peso. Este valor es utilizado posteriormente en el cálculo de los diferentes tipos de métricas. Las métricas presentes en este método son: **Métrica de Costos:** Métrica que determina el costo de desarrollo de software a partir del tiempo que llevará el desarrollo del proyecto, las tarifas asociadas a la cantidad de roles estimados inicialmente por el Jefe de Proyecto y el total de hombres necesarios. **Métrica de Recursos:** Métrica que representa el total de hombres necesarios para el desarrollo del software, obtenidos a partir del esfuerzo y del esfuerzo máximo de desarrollo del tiempo real de desarrollo. **Métrica de Esfuerzo:** Métrica que calcula el esfuerzo de desarrollo a partir de los tiempos dedicados a las etapas. También calcula el esfuerzo máximo de desarrollo teniendo en cuenta el tiempo dedicado a trabajar diariamente en el proyecto. **Métrica de Tiempo:** Métrica que representa el tiempo de desarrollo por cada etapa identificada en el proceso de desarrollo de software. Son porcientos definidos que representan el total de los tiempos dedicados a cada actividad respecto al tiempo total de desarrollo. **Métrica de Tamaño:** Métrica que unifica los requisitos según la complejidad, donde influye el peso definido en cada uno de ellos.

2.2 Especificación de los requisitos

La Especificación de Requisitos de Software (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar, en este caso, de la extensión de la herramienta CASE “Visual Paradigm for UML”. Describe todas las interacciones que tendrán los usuarios con la extensión.

2.2.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y que establecen los comportamientos del sistema. De acuerdo con Ian Sommerville: “Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema” (28).

Requisitos funcionales identificados

RF1. Captar la complejidad de los requisitos funcionales del diagrama.
Descripción: Capturar el valor de la complejidad de los requisitos identificados en el diagrama de requisitos.
Entrada: Diagrama de Requisitos.
Salida: Complejidad de los Requisitos.

RF2. Calcular métricas de Tamaño.
Descripción: Realizar el cálculo de las métricas de tamaño.
Entradas: Total de requisitos de complejidad alta, peso máximo de los requisitos de complejidad alta, total de requisitos de complejidad media, peso máximo de los requisitos de complejidad media, total de requisitos de complejidad baja y peso máximo de los requisitos de complejidad baja.
Salida: Total de Unidades de Desarrollo Ajustados.

RF3. Calcular métricas de Tiempo.
Descripción: Realizar el cálculo del tiempo de desarrollo por cada etapa.
Entradas: Tiempo correspondiente a la etapa de implementación de un requisito, total de unidades de desarrollo ajustados, tiempo correspondiente a la etapa de análisis y diseño, factor de valor agregado, tiempo correspondiente a la etapa de pruebas internas y tiempo correspondiente a la etapa de pruebas de liberación.
Salida: Tiempo Real de Desarrollo.

RF4. Calcular métricas de Esfuerzo.
Descripción: Se realiza el cálculo del esfuerzo de desarrollo y el esfuerzo máximo de desarrollo. Se tendrán en cuenta todos los tiempos dedicados a las etapas de "Análisis y Diseño", "Implementación", "Pruebas Internas" y "Pruebas de Liberación".
Entradas: Tiempo de trabajo diario, cantidad de días laborables a la semana, tiempo límite de solución, tiempo determinado para las de pruebas de aceptación, tiempo real de desarrollo, pruebas piloto, despliegue y soporte.
Salida: Esfuerzo Máximo.

RF5. Calcular métricas de Recursos.
Descripción: Determinar el total de hombres necesarios para el desarrollo del software. Se obtiene a partir del esfuerzo de desarrollo y el esfuerzo máximo de desarrollo en tiempo real.
Entradas: Esfuerzo máximo y esfuerzo.
Salida: Total de Hombres.

RF6. Calcular métricas de Costo.
Descripción: Calcular el costo de desarrollo de software que se necesita a partir del tiempo que se tomará para el desarrollo y las tarifas asociadas a la cantidad de roles clasificados.
Entradas: Tarifas por rol y cantidad de usuarios por rol.
Salida: Costo total de la Aplicación.

RF7. Definir Factor de Valor Agregado.
Descripción: Determina acciones que representan un esfuerzo adicional en el desarrollo del software.
Entradas: Factor obtenido y Factor obtenido asignado a las valoraciones pesimistas.
Salida: Valor agregado.

RF8. Definir Factor de Complejidad Técnica.
Descripción: Se asocia a la complejidad de la tecnología de los requisitos no funcionales del sistema, e influye directamente en el esfuerzo.
Entradas: Cantidad de Requisitos con Reutilización, Cantidad de Requisitos a Desarrollar, Cantidad de Sistemas a Integrar o Legar, Cantidad de Sistemas a Integrar o Legar Desconocidos, Cantidad de

Dispositivos Externos, Cantidad de Dispositivos Externos con Driver o SDK, Cantidad de Estándares y Normas a cumplir, Cantidad de Estándares y Normas específicas.
Salida: Complejidad Técnica.

RF9. Consultar estimación.
Descripción: Consultar la estimación que se realizó del proyecto.
Entrada: Estimación realizada con la extensión.
Salida: Reporte de estimación.

RF10. Generar reporte de estimación en formato PDF.
Descripción: Se generan un documento en formato de PDF con los datos resultantes de la estimación realizada.
Entrada: Estimación realizada con la extensión.
Salida: Reporte de estimación en formato PDF.

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Normalmente están vinculados a requisitos funcionales, es decir, una vez se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser (28).

- **Requisitos de Software:**

RNF1. Se debe tener instalada la herramienta “Visual Paradigm for UML” en su versión 8.0.
RNF2. Se debe tener el JRE (Java Runtime Environment o Entorno en tiempo de ejecución para Java) en su versión 1.6.

- **Requisitos de Hardware:**

RNF3. La capacidad mínima para la memoria RAM (Random Access Memory o Memoria de Acceso Aleatorio) debe ser de 512 MB, aunque se recomienda 1.0 GB.
RNF4. La capacidad mínima de espacio libre en el disco duro debe ser 800 MB.
RNF5. Tarjeta madre (motherboard) Intel Pentium III, compatible con procesador a 1.0 GHz o superior.

- **Restricciones del diseño y la implementación:**

RNF6. Se hace uso de la herramienta CASE “Visual Paradigm for UML” en su versión 8.0 y el IDE NetBeans 7.2.1.

RNF7. El lenguaje de programación que será usado para la implementación es Java 1.6 siguiendo el paradigma de la Programación Orientada a Objeto.

RNF8. El patrón arquitectónico que se debe emplear en el desarrollo es el modelo-vista-controlador.

- **Requisitos de Disponibilidad:**

RNF9. La interfaz debe brindar la posibilidad de ser utilizada por personas que posean conocimientos básicos de informática.

RNF10. Debe ser lo más representativa posible y se podrá hacer uso de la extensión siempre que esté integrada con la herramienta “Visual Paradigm for UML” en su versión 8.0.

RNF11. Las operaciones a realizar por los usuarios deben estar bien descritas, de manera que se puedan entender claramente.

- **Requisitos de Soporte:**

RNF12. El sistema estará acompañado de un documento que guiará la ejecución del usuario teniendo en cuenta cada funcionalidad.

- **Requisitos de Portabilidad:**

RNF13. La extensión, una vez integrada a la herramienta CASE “Visual Paradigm for UML”, podrá ser utilizada en diferentes sistemas operativos por ser ésta una herramienta multiplataforma.

2.3 Modelo de casos de uso

El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. Un caso de uso representa una unidad discreta de interacción entre este y un usuario (humano o máquina). Un modelo de caso de uso es un modelo de las funciones previstas del mismo y su entorno, que sirve como un contrato entre el cliente y los desarrolladores. Su objetivo fundamental es comunicar el comportamiento del sistema al cliente o usuario, representando las relaciones existentes entre actores y casos de uso.

2.3.1 Actores del sistema

Los actores del sistema son los usuarios que interactúan con él y que no son parte del sistema. Incluyen usuarios humanos o sistemas computarizados. En el caso particular de esta extensión no hay sistemas

computarizados que sean actores, los actores siempre van a ser humanos. Estos pueden intercambiar información con el sistema y ser un recipiente pasivo de información. Representan el rol que juega una o varias personas, un equipo o un sistema automatizado (29). En este caso es el planificador. Un actor es un usuario del sistema que desempeña alguna porción de trabajo que es de valor para el negocio, a través de la realización de un caso de uso. 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 (29).

Nombre del actor	Descripción
Planificador	Es el que interactúa con la herramienta CASE “Visual Paradigm for UML” para realizar la estimación mediante la extensión. Luego éste puede consultar o exportar el reporte de la estimación realizada.

Tabla 8: Actor del sistema

2.3.2 Diagrama de casos de uso

Un diagrama de casos de uso representa gráficamente a los procesos y su interacción con los actores. Un caso de uso es una unidad de trabajo significativo. Cada uno de ellos tiene una descripción que especifica la funcionalidad que se incorporará al sistema propuesto. Pueden 'incluir' o 'extender' la funcionalidad de otro caso de uso, con su propio comportamiento. Los mismos deben comunicarse al menos con un actor u otro caso de uso. No deben aparecer aislados y sin relacionar, esto indica error en el diagrama modelado o en los requisitos planteados. En el caso específico de esta extensión se agruparon los diez requisitos funcionales identificados en dos casos de uso: Realizar Estimación y Consultar Estimación. Además, se identificó como actor del sistema al planificador de proyectos (30).

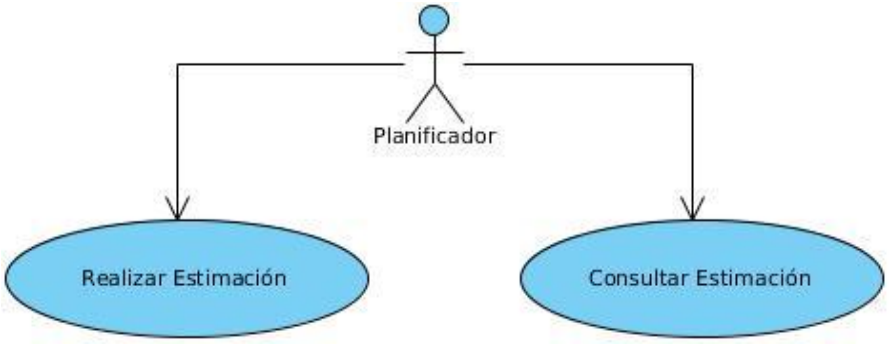


Imagen 3: Diagrama de Casos de Uso

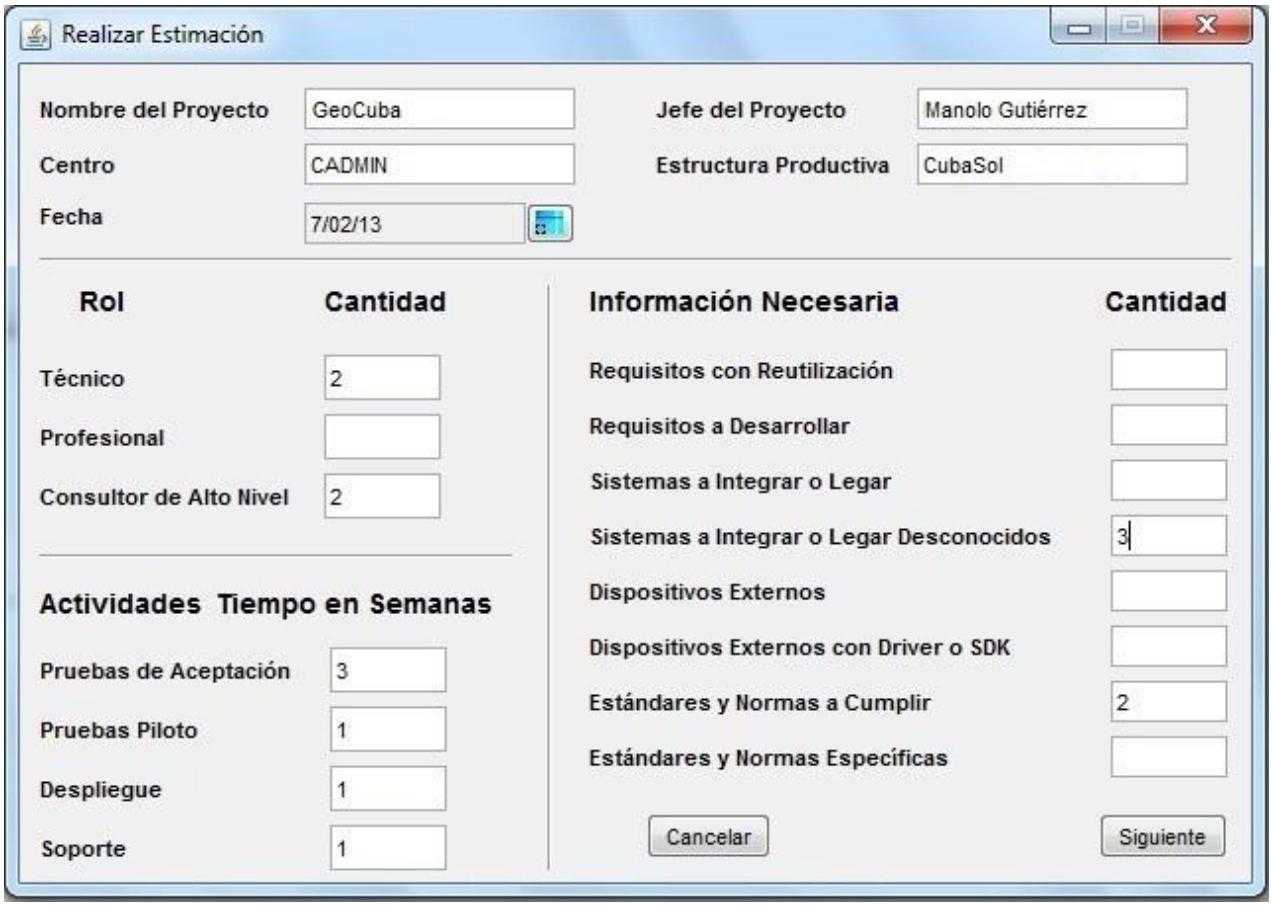
Descripción textual del caso de uso más importante

Luego de modelar el diagrama de casos de uso del sistema, se realizó la descripción textual de cada uno de los casos de uso que forman parte del diagrama. Dichas descripciones se encuentran reflejadas en el artefacto Modelo de Sistema (Ver Expediente de Proyecto). A continuación se muestra como ejemplo la descripción textual del caso de uso "Realizar Estimación".

<table>
<tr><td>Caso de Uso:</td><td colspan="2">Realizar estimación</td></tr>
<tr><td>Actores:</td><td colspan="2">Planificador</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de uso inicia cuando el usuario presiona clic derecho en el diagrama de requisitos modelado en la herramienta CASE "Visual Paradigm for UML" y selecciona la opción "Realizar Estimación". El sistema muestra la interfaz "Realizar Estimación" que posee los campos que el usuario debe llenar.</td></tr>
<tr><td>Precondiciones:</td><td colspan="2">Que el usuario haya modelado el diagrama de requisitos en la herramienta.</td></tr>
<tr><td>Referencias</td><td colspan="2">RF1, RF2, RF3, RF4, RF5, RF6, RF7, RF8, RF10.</td></tr>
<tr><td>Prioridad</td><td colspan="2">Alta</td></tr>
<tr><td colspan="3">Flujo Normal de Eventos</td></tr>
<tr><td colspan="3">Sección 1 "Realizar estimación"</td></tr>
<tr><td colspan="2">Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2">1. El usuario presiona clic derecho en el diagrama de requisitos modelado en la herramienta CASE "Visual Paradigm for UML" y selecciona la opción "Realizar Estimación".</td><td>2. El sistema muestra la interfaz "Realizar Estimación" que indica los datos que el usuario debe llenar.</td></tr>
<tr><td colspan="2">3. El usuario completa los datos.</td><td>4. El sistema verifica si no existen campos vacíos y valida que los datos estén correctos.
En caso de detectar algún error ver el</td></tr>
</table>

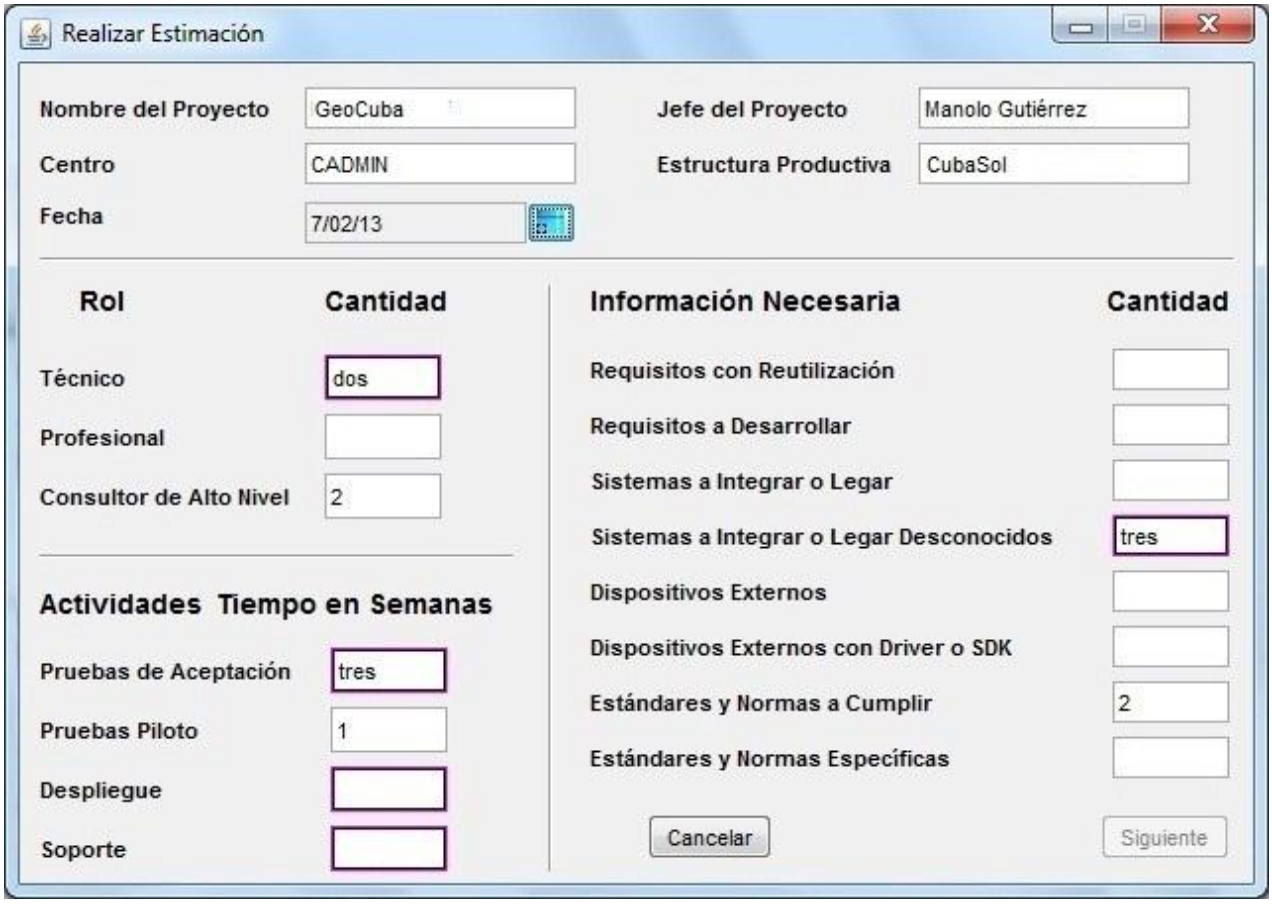
	flujo alterno. Si no se detectan errores el sistema habilita el botón **"Siguiente"**.

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	4.1. Si existen campos vacíos o los datos son incorrectos, el sistema señala en color morado el o los campos vacíos, e inhabilita el botón **"Siguiente"**.

Prototipo de Interfaz

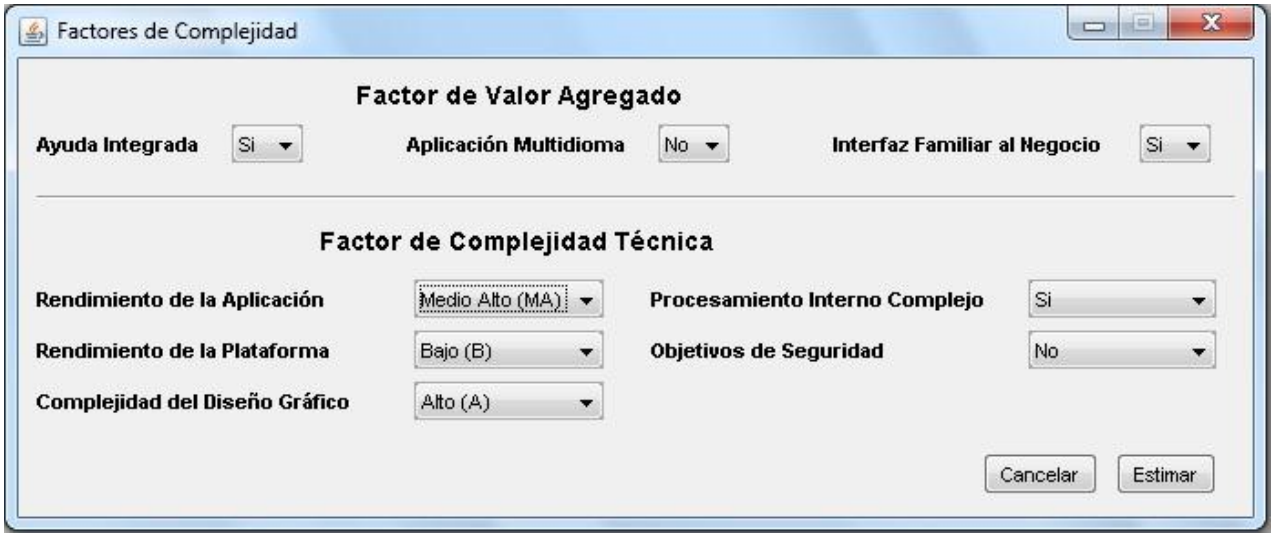


Poscondiciones	

Sección 2 "Calcular Factores de Complejidad"	
Acción del Actor	**Respuesta del Sistema**
1. El usuario selecciona la opción **"Siguiente"**.	2. El sistema muestra la interfaz **"Factores de Complejidad"** que contiene los datos que el usuario debe llenar relacionados con los factores de complejidad necesarios para el cálculo de la estimación.

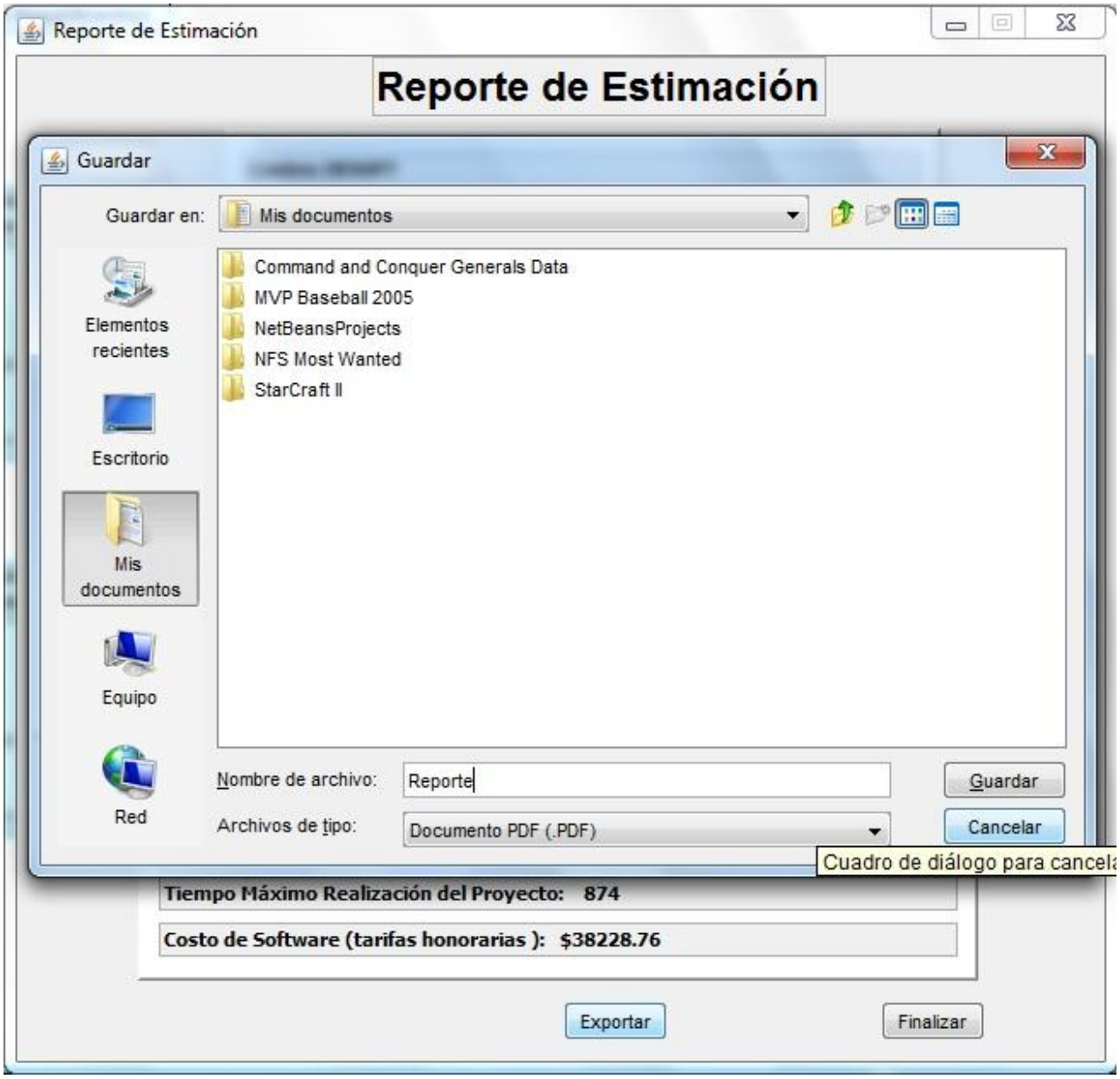
3. El usuario asigna valores a cada uno de los factores de complejidad y selecciona la opción **"Estimar"**.	4. El sistema calcula los factores de complejidad con los datos obtenidos de la interfaz **"Factores de Complejidad".**
	5. El sistema captura los datos del diagrama de requisitos y realiza el cálculo de las métricas de tiempo, costo, esfuerzo, tamaño y recursos.
	6. El sistema realiza la estimación con los datos obtenidos del cálculo de los factores de complejidad y las métricas.
	7. El sistema muestra el **"Reporte de Estimación".**
8. El usuario selecciona la opción: **"Finalizar"** o **"Exportar"**.	9. Si el usuario selecciona la opción **"Finalizar"** el sistema cierra la ventana terminando así el caso de uso. Si no fue seleccionada dicha opción ver flujo alterno.

Prototipo de Interfaz



Flujo Alterno	
Acción del Actor	**Respuesta del Sistema**
	9.1. Si el usuario selecciona la opción **"Exportar"** el sistema muestra una ventana de diálogo brindando la posibilidad al usuario de seleccionar la dirección donde desea guardar el reporte de la estimación.

Prototipo de Interfaz



Poscondiciones	Se realiza la estimación y se genera el reporte de la misma.

Tabla 9: Descripción del caso de uso "Realizar Estimación"

2.3.3 Matriz de trazabilidad

La matriz de trazabilidad es una técnica que relaciona los requisitos funcionales a los diferentes elementos del desarrollo. Es una herramienta que se utiliza para saber qué requisitos quedan cubiertos por casos de uso y de esta forma garantizar que todos los elementos para el desarrollo de la extensión sean ejecutados correctamente (10).

Casos de Uso	Requisitos Funcionales									
	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	RF10
CU1- Realizar Estimación	X	X	X	X	X	X	X	X		X
CU2- Consultar Estimación									X	X

Tabla 10: Matriz de Trazabilidad.

2.4 Modelo de diseño

El modelo de diseño es una simplificación utilizada para comprender mejor el problema, dando lugar a la creación de una solución para el problema una vez que se analice. Anticipa y guía el proceso de producción. Define una solución de software que satisface los requisitos especificados en el análisis. Incorpora nuevos artefactos: clases, atributos y operaciones para las clases; además se tendrá en cuenta la plataforma tecnológica concreta sobre la que debe construirse el sistema software (31).

2.4.1 Diagrama de clases del diseño

El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y las interfaces en una aplicación. A diferencia del modelo conceptual, un diagrama de este tipo contiene las definiciones de las entidades de software en vez de conceptos del mundo real (31). Para la elaboración del modelo de diseño fueron modelados dos diagramas de clases del diseño, cada uno de ellos en correspondencia con un caso de uso del sistema. Dichos diagramas y sus descripciones se encuentran reflejados en el artefacto Modelo de Diseño (Ver Expediente de Proyecto). Las clases contenidas en cada uno de los paquetes que conforman los diagramas pueden ser agrupadas teniendo en cuenta el patrón arquitectónico MVC[7] aplicado a la extensión.

[7]MVC: Modelo Vista Controlador

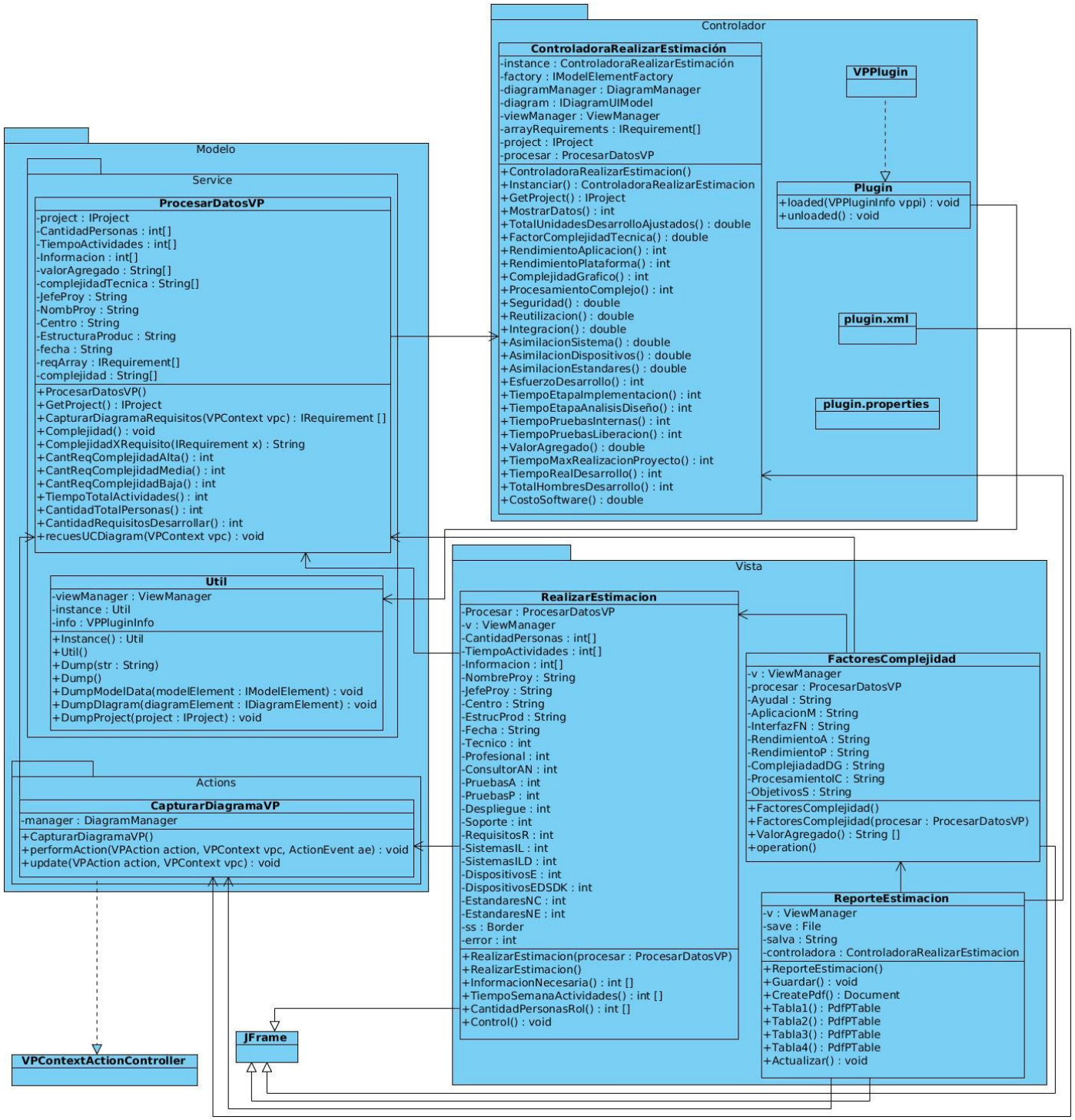


Imagen 4: Diagrama de Clases del Diseño del caso de uso "Realizar Estimación"

2.4.2 Descripción de las clases más relevantes del diagrama de clases del diseño

A partir de los diagramas de clases del diseño, se realizó la descripción de las clases relevantes que forman parte de cada uno de estos. Dichas descripciones se encuentran reflejadas en el artefacto Modelo de Diseño (Ver Expediente de Proyecto). A continuación se muestran las descripciones de las clases principales del diagrama de clases del diseño correspondiente al caso de uso **"Realizar Estimación"**:

No.	Clase	Tipo de clase	Descripción	Relación	Tipo (relación)
1	Controladora Realizar Estimación	Controladora	Clase que contiene los métodos fundamentales para el cálculo de la estimación y el desarrollo de la extensión.	4, 10	Asociación
2	Plugin	Controladora	Permite generar la extensión con los atributos correspondientes.	5	Asociación
3	plugin.xml	Controladora	Es la clase que muestra las opciones **Realizar Estimación** y **Consultar Estimación** en el diagrama de requisitos realizado en la herramienta CASE "Visual Paradigm for UML".	6, 7	Asociación
4	Procesar DatosVP	Modelo	Clase que contiene métodos auxiliares que facilitan la comunicación entre la clase controladora y las clases interfaz, así como la interpretación de los datos recibidos de estas.	1, 6, 8, 9	Asociación
5	Util	Modelo	Captura las propiedades de la extensión.	2	Asociación

6	Capturar DiagramaVP	Modelo	Es la clase encargada de capturar los datos del diagrama para que sean interpretados y procesados por la clase 4.	4, 8, 12	Asociación
7	Clase Consultar Estimacion	Modelo	Gestiona el reporte de la estimación realizada.	11	Asociación
8	Realizar Estimacion	Interfaz	Es la clase que contiene los elementos que caracterizan al sistema al cual se le realizará la estimación.	4, 6, 9	Asociación
9	Factores Complejidad	Interfaz	Esta clase posee los factores de complejidad que influyen en las métricas.	4, 8, 10	Asociación
10	Reporte Estimacion	Interfaz	Es la clase que se encarga de mostrar el reporte de estimación.	1, 9	Asociación
11	Consultar Estimacion	Interfaz	Esta clase muestra el reporte de la última estimación realizada a este sistema.	7	Asociación
12	VPContext Action Controller	Modelo	Es la clase que brinda el Open API por defecto, que permite visualizar y ejecutar las acciones del usuario en la herramienta.	6	Realización

Tabla 11: Descripción de las clases principales de la extensión

2.5 Diagrama de secuencia

El UML provee un medio gráfico para representar la interacción entre los objetos a lo largo del tiempo en los diagramas de secuencia. Éstos muestran típicamente a un usuario o a un actor, así como los objetos y componentes con los que interactúa durante la ejecución de un caso de uso. Un diagrama de secuencia representa típicamente un único escenario de caso de uso o flujo de eventos. El diagrama de secuencia explica gráficamente las interacciones existentes entre las instancias. De ellos se identifican las clases de software que intervienen en la solución y sus métodos (32). Su punto de partida es el cumplimiento de las pos-condiciones de los contratos de una operación. Para mejorar la calidad de su diseño, es posible aplicar patrones, principios y expresiones codificadas. Los diagramas de secuencia son una vía excelente para documentar los escenarios de uso, para capturar los objetos necesarios de manera temprana en el análisis y para verificar el uso de los objetos más tarde en el diseño. Muestran el flujo de mensajes de un objeto a otro y, como tales, representan los métodos y los eventos soportados por un(a) objeto o clase (19).

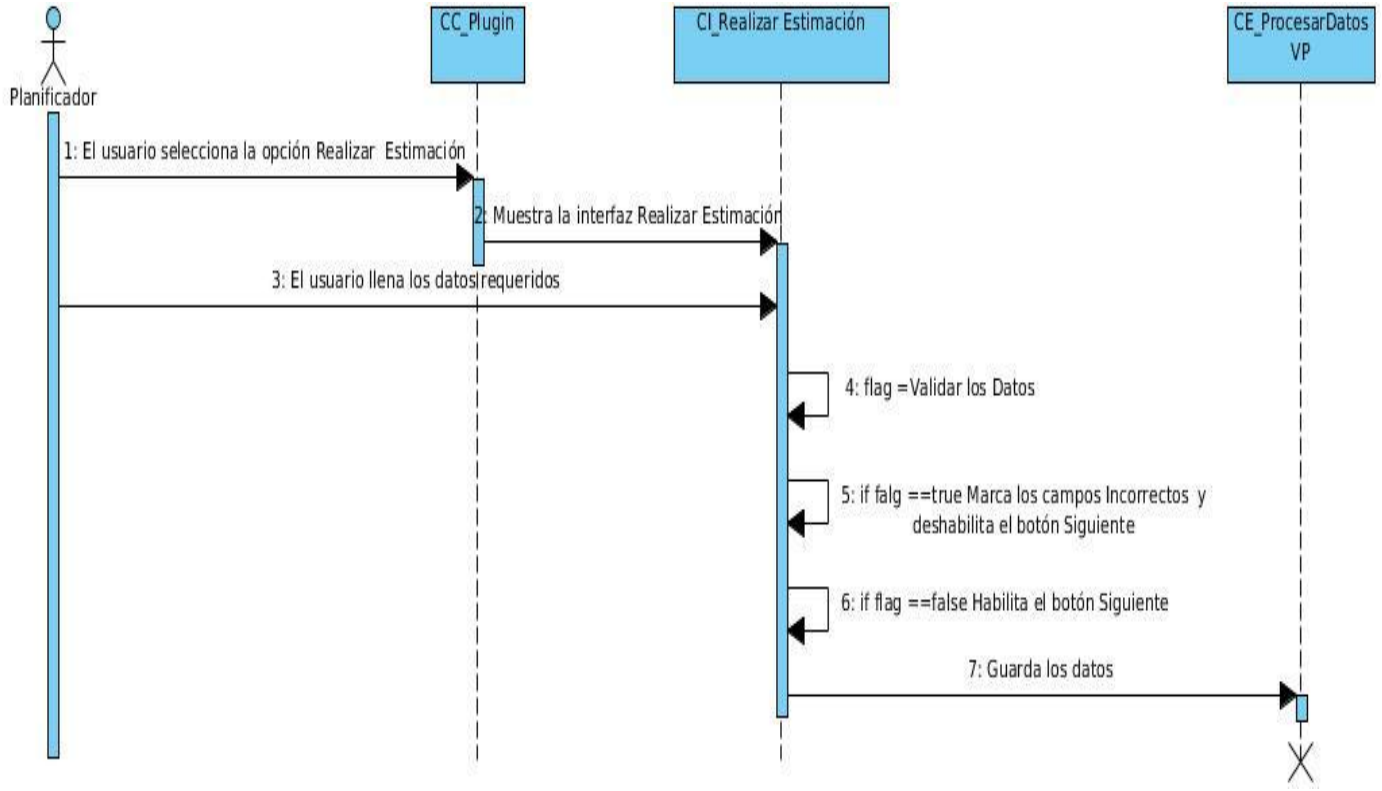


Imagen 5: Diagrama de Secuencia del Caso de Uso "Realizar Estimación" sección 1

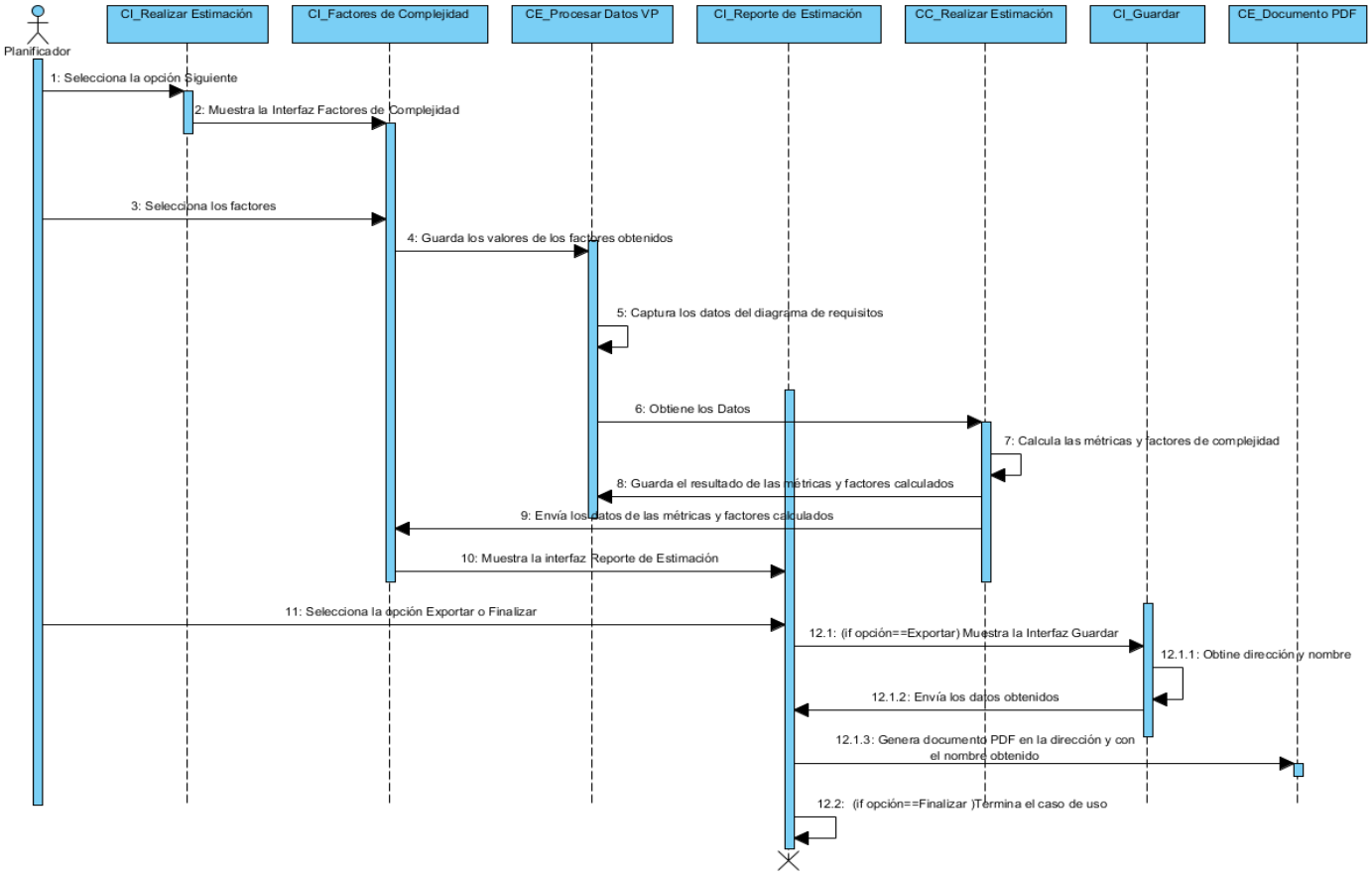


Imagen 6: Diagrama de Secuencia del Caso de Uso "Realizar Estimación" sección 2

2.6 Patrones utilizados

Los patrones son un amplio repertorio de principios generales basados en la experiencia, que guían la creación de un software. Cada patrón describe un problema que ocurre una y otra vez en un entorno, para describir después el núcleo de la solución a ese problema. De esa manera dicha solución puede ser usada muchas veces sin hacerlo siquiera dos veces de la misma forma. Existen dos tipos de patrones: los patrones de diseño y los de arquitectura. Los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones), con las que construir sistemas de software (33). Los patrones de arquitectura expresan un esquema organizativo estructural fundamental para sistemas de software. En el desarrollo de la extensión fueron puestos en práctica algunos de estos. Entre ellos destacan los patrones Modelo-Vista-Controlador, Gang Of Four (GOF por sus siglas en inglés) y General Responsibility Assignment Software Patterns (GRASP por sus siglas en inglés), respectivamente (33).

2.6.1 Patrón Arquitectónico: Modelo-Vista-Controlador

El patrón de arquitectura Modelo-Vista-Controlador permite realizar la programación multicapa, separando en tres componentes distintos los datos de una aplicación, la interfaz del usuario y la lógica de control. Para definir la arquitectura de la extensión propuesta, resulta fundamental regirse por los elementos arquitectónicos definidos en el API de la herramienta CASE “Visual Paradigm for UML”. Esta propone una arquitectura basada en dicho patrón; 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 (34):

Modelo: Representa la información con la que trabaja la aplicación, o sea, su lógica de negocio. Es el objeto que maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo. Es el responsable de administrar el comportamiento del dominio de la aplicación, empleando para ello las acciones contenidas en el openapi.jar.

Vista: Es el objeto que maneja la presentación visual de los datos representados por el modelo. Genera una representación visual del modelo, interactuando con él a través de una referencia al mismo y muestra los datos al usuario. Es la responsable de la visualización de la información, a través de los diferentes tipos de diagramas que provee la herramienta CASE “Visual Paradigm for UML” y las interfaces mostradas al desarrollador.

Controlador: Es el encargado de procesar las interacciones del usuario y ejecuta los cambios adecuados en el modelo o en la vista. 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), procesan los datos y muestran el resultado en una clase (Vista). Proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo (33).

2.6.2 Patrones de diseño GoF

Los patrones GoF se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento:

- **Creacionales:** Los patrones creacionales abstraen el proceso de creación de instancias y ocultan los detalles de cómo los objetos son creados o inicializados.

- **Estructurales:** Los patrones estructurales se ocupan de cómo las clases y objetos se combinan para formar grandes estructuras y proporcionar nuevas funcionalidades.
- **Comportamiento:** Los patrones de comportamiento están relacionados con los algoritmos y la asignación de responsabilidades entre los objetos. Son utilizados para organizar, manejar y combinar comportamientos (33).

Factory Method (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 (35). 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 (35). Este patrón resulta útil para la etapa de implementación de la extensión, 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 de la extensión. Su utilización se evidencia en la clase ControladoraRealizarEstimación.java.

Singleton (Solitario)

Es de tipo creacional, a nivel de objetos. Su propósito es garantizar que una clase 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 (36). El patrón singleton o solitario es implementado por la clase ControladoraRealizarEstimación.java permitiendo la utilización de una sola instancia de dicha clase.

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, por lo que se facilitan el paralelismo y la concurrencia. 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 (37). Este patrón se encuentra implementado en la clase ProcesarDatosVP.

2.6.3 Patrones de diseño GRASP

Los patrones GRASP representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. Resaltan la importancia de captar principios para la programación, a la hora de diseñar eficazmente un software orientado a objetos (38).

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 (38). Esto se evidencia en la clase ProcesarDatosVP.java.

Creador

Este patrón asigna responsabilidades relacionadas con la creación de objetos. En la solución propuesta se utiliza al asignarle la responsabilidad a las clases controladoras de crear un objeto de las clases modelos para el posterior acceso a las funciones implementadas en el modelo (38). Esto se evidencia dentro del marco de trabajo de la clase CapturarDiagramaVP.java. En dicha clase 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 negocio debe estar separada de la capa de presentación, para aumentar la reutilización de código y a la vez tener un mayor control (38). Se pone de manifiesto en la clase ControladoraRealizarEstimación.java.

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, tenga la mínima repercusión posible en el resto de las 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 (38). En la implementación de la extensión es aplicado este patrón en la mayoría de las clases. Ejemplo de la implementación de este patrón se evidencia entre las clases ControladoraRealizarEstimacion,

ProcesarDatosVP y RealizarEstimacion; lo que propicia 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. Ejemplos: las clases CapturarDiagramaVP.java y ControladoraRealizarEstimación.java. Estas clases 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 (38). Para la implementación y funcionalidad de la extensión se implementan patrones arquitectónicos y de diseño agilizando el proceso de implementación.

2.7 Diagrama de despliegue

Un diagrama de despliegue constituye una representación física de las relaciones que existen entre los componentes de hardware y software en el sistema. A través de este diagramase representa la configuración de los elementos de procesamiento y los componentes de software en tiempo de ejecución. Solo los componentes que son utilizados en tiempo de compilación deben mostrarse en el diagrama de despliegue (39). Los componentes de la extensión son compilados por la herramienta Visual Paradigm for UML una vez iniciada la aplicación, formando parte de un único nodo físico de procesamiento. Dada esta particularidad no resulta efectivo realizar el modelado del diagrama de despliegue.

Conclusiones parciales

En este capítulo se elaboró el modelo de dominio, facilitando una mejor compresión dentro del contexto del sistema. Se identificaron diez requisitos funcionales que fueron agrupados en dos casos de uso y trece requisitos no funcionales, permitiendo definir las características y las condiciones que debe poseer la extensión. Se detallaron los patrones de diseño que serán empleados para el proceso de desarrollo de la extensión, lo que posibilitó asignar las responsabilidades adecuadas a cada una de las clases. Se describió además el patrón arquitectónico Modelo-Vista-Controlador utilizado, que permitió una separación de conceptos y un desarrollo estructurado del sistema. Además, gracias al análisis de las relaciones que existen entre los componentes de hardware y software en el sistema, se concluyó que no es necesario confeccionar el diagrama de despliegue debido a que todos los elementos de la extensión están contenidos en un único nodo físico.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

Introducción

En este capítulo se realizará la implementación de la extensión, así como las pruebas funcionales de esta para comprobar si se corresponde el sistema desarrollado con los requisitos de software anteriormente definidos. Se define el modelo de implementación que muestra el diseño de la solución, así como el diagrama de componentes de la extensión. Se especifican, además, las pruebas realizadas a la extensión, con el objetivo de comprobar sus funcionalidades en los diferentes escenarios y de esta forma verificar en todos los casos que los resultados de las pruebas sean los esperados.

3.1 Modelo de implementación

En el modelo de implementación se muestra cómo se implementan los elementos del modelo de diseño en términos de componentes. Se describen los componentes a construir y su organización en nodos físicos. Estos comprenden un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema (40).

3.1.1 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. Incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables o paquetes. Este diagrama representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos. Son utilizados para modelar la vista estática y dinámica de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. En dicho diagrama se muestran los elementos de diseño de un sistema de software. Permite visualizar con mayor facilidad la estructura general del sistema. Además, muestra el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces (41).

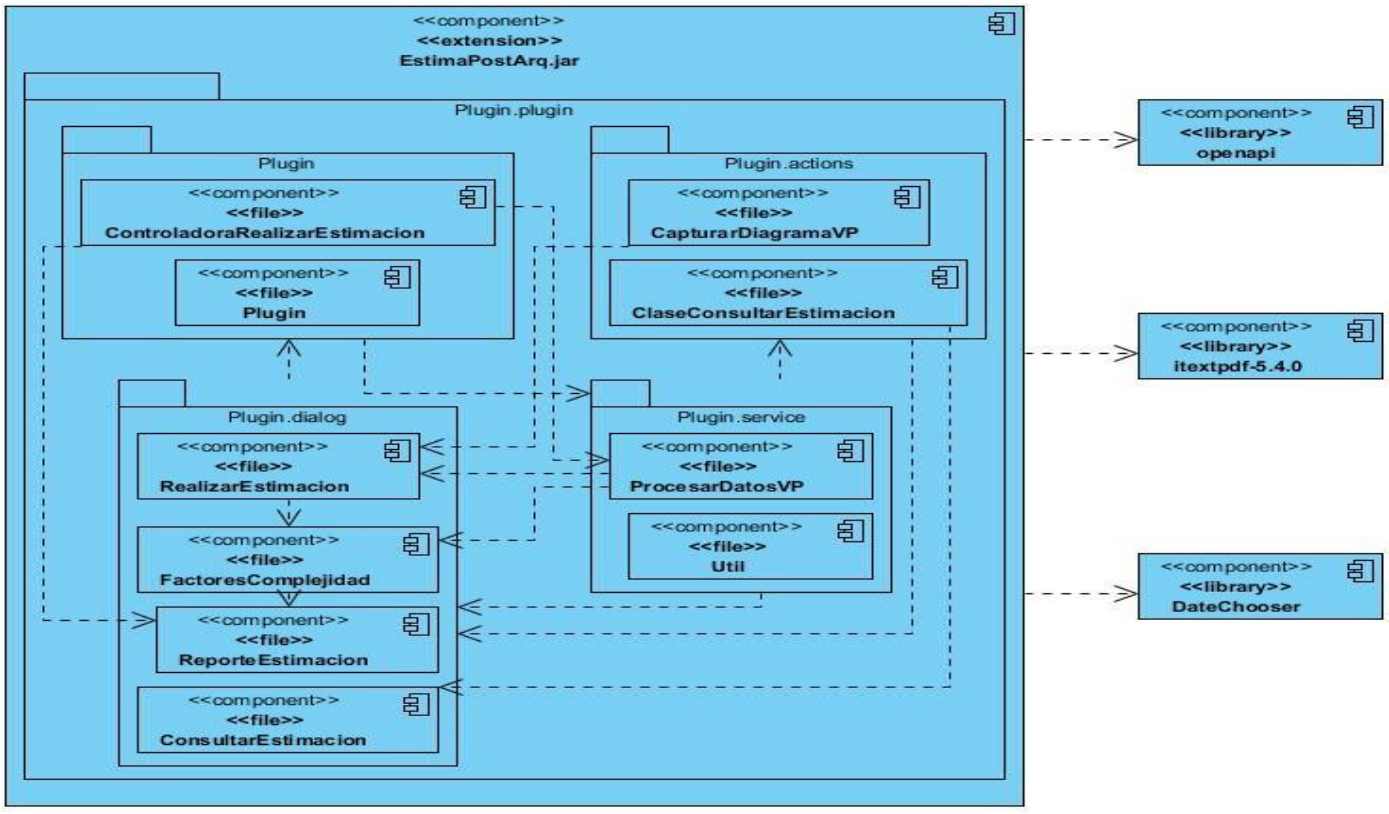


Imagen 7: Diagrama de Componentes

3.2 Estándares de codificación

Los estándares de codificación son reglas que se siguen para la escritura del código fuente. De esta manera a otros programadores les resulta más factible entender el código: identificar las clases, las variables, las funciones o métodos. Los estándares reúnen técnicas de codificación sólidas y un conjunto de buenas prácticas de programación. Estos son de gran importancia para la calidad de un producto de software y obtener un buen rendimiento facilitando:

- El mantenimiento del sistema. Aumenta la facilidad con que el sistema puede modificarse: añadirle nuevas características, modificar las ya existentes, depurar errores y mejorar el rendimiento.
- Que se genere un código de programación legible, lo que permite una mayor comprensión del sistema por parte de los programadores.
- La corrección de errores en el sistema.
- El entendimiento y manejo del sistema.

3.2.1 Estilos de codificación empleados

Para la implementación de la extensión se aplicaron los estándares de codificación de Java dentro de los que se encuentran (42):

- **Organización de los ficheros**

Serán evitados los ficheros de gran tamaño que contengan más de 1500 líneas de código, pues en ocasiones el tamaño excesivo provoca que la clase no encapsule un comportamiento claramente definido, albergando una gran cantidad de métodos que realizan tareas funcional o conceptualmente heterogéneas (42).

- **Tamaño y organización de las líneas de código**

La longitud de línea no debe superar los 80 caracteres. En caso de que una expresión ocupe más de una línea, esta se podrá romper o dividir tras una coma o antes de un operador y la nueva línea debe estar alineada con el inicio de la expresión al mismo nivel que la línea anterior (42).

- **Indentación**

Se deben emplear cuatro espacios como unidad de indentación. La construcción exacta de la indentación (espacios en blanco contra tabuladores) no se especifica. Los tabuladores deben ser exactamente cada ocho espacios (42).

- **Declaraciones**

Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso. En las declaraciones de los métodos, no debe incluirse ningún espacio entre el nombre del método y el paréntesis inicial del listado de parámetros (42).

Los métodos se separarán entre sí mediante una línea en blanco. Se recomienda una declaración por línea, ya que facilita los comentarios. Poner las declaraciones solo al principio de los bloques (un bloque es cualquier código encerrado por llaves "{" y "}".) No esperar al primer uso para declararlas; puede confundir a programadores no familiarizados con el sistema y limitar la portabilidad del código dentro de su ámbito de visibilidad (42).

- **Sentencias**

Cada línea debe contener a lo sumo una sentencia. En el caso de ser sentencias compuestas, que contienen listas de sentencias encerradas entre llaves "{sentencias}", deben (42):

- Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
- La llave de apertura se debe poner al final de la línea que comienza la sentencia compuesta; la llave de cierre debe empezar una nueva línea y ser indentada al mismo nivel que el principio de la sentencia compuesta.
- Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias if-else o for. Esto hace más sencillo añadir sentencias sin incluir bugs accidentalmente por olvidar las llaves.
- El caracter inicio de bloque debe situarse al final de la línea que inicia el bloque. El caracter final de bloque debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer caracter de dicho bloque. Todas las sentencias de un bloque deben encerrarse entre llaves, aunque el bloque conste de una única sentencia.

- **Espacios en blanco**

Las líneas en blanco mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas.

Se deben usar siempre dos líneas en blanco en las siguientes circunstancias:

- Entre las secciones de un fichero fuente.
- Entre las definiciones de clases e interfaces.

Se debe usar siempre una línea en blanco en las siguientes circunstancias (42):

- Entre métodos.
- Entre las variables locales de un método y su primera sentencia.
- Antes de un comentario de bloque o de un comentario de una línea.
- Entre las distintas secciones lógicas de un método para facilitar la lectura.

- **Nomenclatura de identificadores**

Paquetes: Los nombres de los paquetes se escribirán siempre en letras minúsculas para evitar que entren en conflicto con los nombres de clases e interfaces (42).

Clases e interfaces: Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivos. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).

Métodos: Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Variables: Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres como guión bajo "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje (42).

3.2.2 Buenas prácticas de programación

- **Visibilidad de atributos de instancia y de clase:** Los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas; en tales casos serán declarados como protegidos. El acceso a los atributos de una clase se realizará por medio de los métodos "get" y "set" correspondientes.
- **Asignación sobre variables:** Se deben evitar las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia, ya que dichas sentencias suelen ser difíciles de leer. No se deben utilizar asignaciones embebidas o anidadas.
- **Paréntesis:** Se deben utilizar paréntesis en expresiones que incluyan distintos tipos de operadores para evitar problemas de precedencia de operadores (42).

3.2.3 Ejemplos de Código Fuente

El código implementado de manera general manifiesta funciones de captura y selección de información. Muestra el reporte con los resultados de los cálculos realizados utilizando los datos recogidos. Exporta en formato .pdf el reporte elaborado. La ejecución de estas funciones se facilita mediante el uso de interfaces de usuario y librerías proporcionadas por Java.

Un ejemplo del código es el método **CapturarDiagramaRequisitos**(VPContex vpc), encargado de capturar los elementos del diagrama de requisitos modelado en la herramienta CASE Visual Paradigm for UML, almacenándolos en una lista. Una vez en la lista, se podrán separar las clases que constituyen requisitos para tratar de manera individual sus propiedades, específicamente el atributo complejidad.

```
public IRequirement[] CapturarDiagramaRequisitos(VPContext vpc) {
    int pos = 0;
    LinkedList<IRequirement> reqList = new LinkedList<IRequirement>();
    IDiagramElement[] elem = vpc.getDiagram().toDiagramElementArray();
    for (int j = 0; j < elem.length; j++) {
        IModelElement element = elem[j].getModelElement();
        if ((element.getModelType().equals(IModelElementFactory.MODEL_TYPE_REQUIREMENT))) {
            reqList.add((IRequirement) element);
        }
    }
    ReqArray = new IRequirement[reqList.size()];
    Iterator<IRequirement> iterator = reqList.iterator();
    while (iterator.hasNext()) {
        ReqArray[pos] = iterator.next();
        pos++;
    }
    Complejidad();
    return ReqArray;
    //Capturando la información de los requisitos
}
```

Imagen 8: Ejemplo de código fuente, método CapturarDiagramaRequisitos(VPContext vpc)

Otro ejemplo lo constituyen, en este caso, los métodos **ComplejidadXRequisito(Irequirement x)** y **Complejidad()**. Estos métodos recorren el arreglo de requisitos resultante del método antes mencionado para formar una lista con los valores de los atributos que especifican en ellos. Se analizan cada uno de los elementos de los requisitos almacenados individualmente. En este caso, el atributo “complejidad” es el que se utiliza para realizar la estimación. El valor adquirido por este atributo en cada uno de los requisitos definidos es almacenado en un arreglo para su posterior utilización.

```
public void Complejidad() {
    Complejidad = new String[ReqArray.length];
    for (int i = 0; i < ReqArray.length; i++) {
        Complejidad[i] = ComplejidadXRequisito(ReqArray[i]);
    }
}

public String ComplejidadXRequisito(IRequirement x) {
    ITaggedValue[] arr = x.getTaggedValues().toTaggedValueArray();
    for (int i = 0; i < arr.length; i++) {
        if (arr[i].getName().equals("complejidad")) {
            return arr[i].getValueAsText();
        }
    }
    return "";
}
```

Imagen 9: Ejemplo de código fuente, método Complejidad()

Los métodos ejemplificados anteriormente se encuentran situados en la clase ProcesarDatosVP; en la cual la información es procesada para la realización de cálculos en métodos auxiliares. Esta información se utiliza posteriormente en la clase ControladoraRealizarEstimacion para el cálculo final de la estimación.

3.2.4 Interfaces Principales de la Extensión

Para el diseño de la extensión se tuvo en cuenta la interfaz sencilla y amigable correspondiente con la herramienta a extender. Se elaboraron de manera fácil y agradable para el usuario de manera tal que no resulte engorroso el trabajo con la misma. La extensión posee tres interfaces principales que corresponden: dos la captura de información (Realizar Estimación y Factores de Complejidad) y una a la muestra del resultado de la estimación realizada (Reporte de Estimación). Además, se puede mencionar la interfaz relacionada con la funcionalidad de exportar en formato .pdf el resultado de la estimación realizada. A continuación se muestran algunas de las interfaces mencionadas.

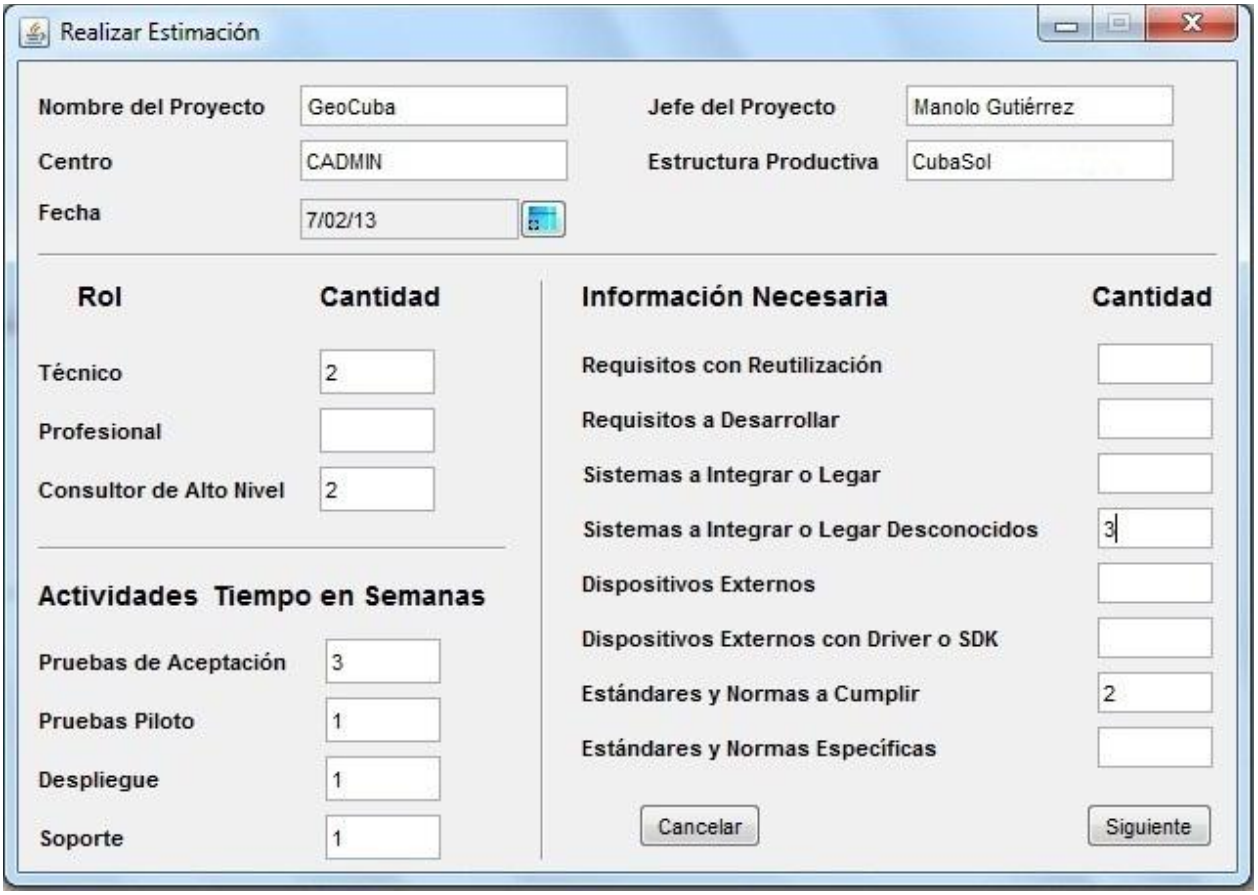


Imagen 10: Interfaz "Realizar Estimación"

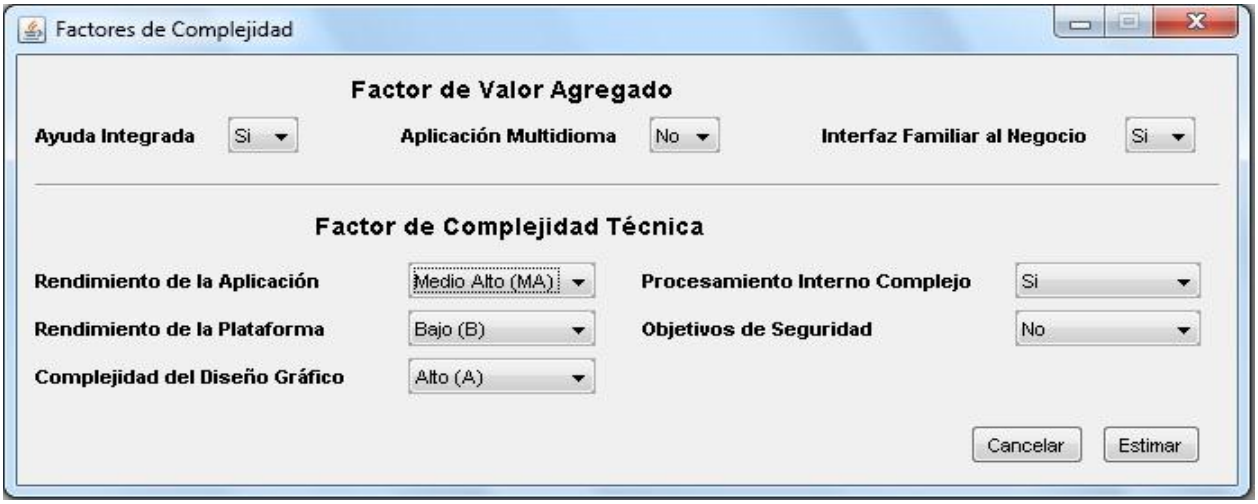


Imagen 11: Interfaz "Factores de Complejidad"

Imagen 12: Interfaz "Reporte de Estimación"

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 de ejecución de un programa con la intención de descubrir un error, que se enfoca en la lógica interna del software y sus funciones externas (43). Tiene éxito si descubre un error no detectado hasta entonces. Además, esta etapa implica:

- 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 (43).

Para comprobar que la extensión funcione de manera correcta, se realizan pruebas de desarrollador, correspondientes al primer nivel de prueba, y que son diseñadas e implementadas por el equipo de desarrollo. Dentro de la pruebas de desarrollador se realizan las pruebas funcionales, que ejecutan a cada una de las funcionalidades para verificar que se cumplan los requisitos establecidos. Estas incluyen la navegación, entrada de datos, procesamiento y obtención de resultados, enfocándose en los requisitos funcionales y casos de uso. Dentro de los métodos que utiliza este tipo de prueba se encuentra el método de Caja Negra, que se seleccionó para aplicarlo a la extensión.

3.1.1 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 se conocen como pruebas de comportamiento, que 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 (44). Para diseñar los casos de prueba de caja negra se utilizó la técnica de 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 (44). Estas pruebas se

encuentran reflejadas en los artefactos: Diseño de Casos de Prueba del Caso de Uso Realizar Estimación y Diseño de Casos de Prueba del Caso de Uso Consultar Estimación (ver Expediente de Proyecto)

Casos de pruebas

Los casos de prueba se definen a partir de las funcionalidades descritas en los casos de uso, validando el correcto funcionamiento de cada uno de ellos. A través de los mismos se especifican los escenarios, sus secciones y variables involucradas en cada proceso. De esta forma se detalla la respuesta del sistema ante cada valor introducido por el usuario, ya sea correcto, incorrecto o nulo. Con el fin de representar los casos de prueba, se utiliza una tabla donde se desglosan las funcionalidades en secciones y a su vez en escenarios, para hacer más fructífera la ejecución de las pruebas (45). En materia del caso de uso más significativo, la tabla antes mencionada puede observarse en el artefacto Diseño de Casos de Prueba del Caso de Uso “Realizar Estimación” (ver Expediente de Proyecto).

En las iteraciones realizadas se aplicaron pruebas al sistema comprobando el correcto funcionamiento de los casos de uso "Realizar Estimación" y "Consultar Estimación". Se comprobó, entre otras funcionalidades, que los datos introducidos por el usuario estuvieran correctos para poder acceder a la siguiente interfaz. Para ello se verificó que no se le permitiera al usuario presionar el botón "Siguiente" mientras existieran campos obligatorios vacíos o datos incorrectos. Para el caso de uso "Consultar Estimación", no existían variables involucradas por lo no fue necesaria la validación a través del método partición de equivalencia. Se comprobó además que el sistema permitiera al usuario consultar la estimación y generar el reporte en un documento con formato .pdf.

3.4 Resultados de las pruebas

Se realizaron pruebas funcionales a la aplicación, utilizando el método de caja negra, aplicando la técnica partición de equivalencia. En el caso de uso Realizar Estimación, para validar el correcto funcionamiento de la extensión fueron realizadas tres iteraciones de pruebas. Se detectaron seis no conformidades en la primera iteración y dos en la segunda. Estas fueron resueltas progresivamente, permitiendo que en la tercera iteración no fueran detectadas no conformidades. En la siguiente gráfica se muestra un resumen de los resultados obtenidos:

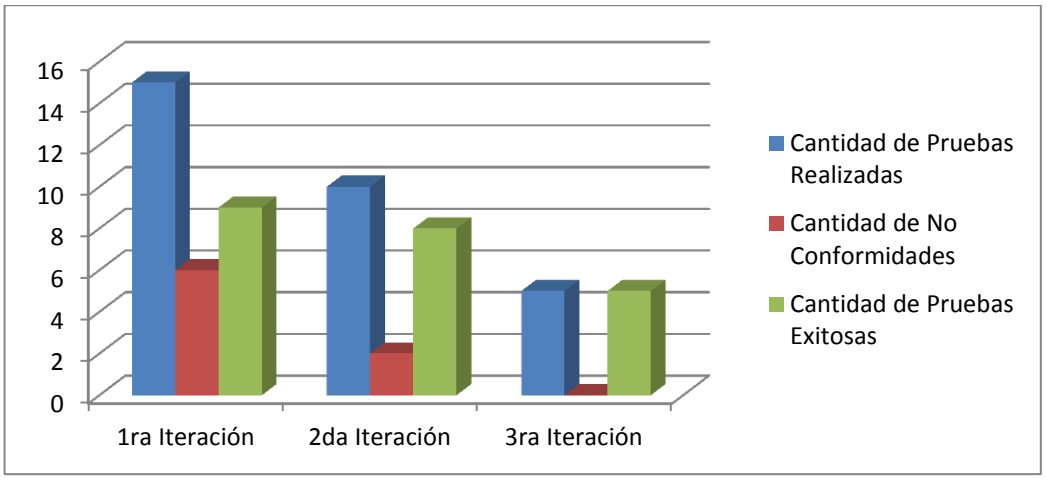


Imagen 13: Resultados de las pruebas realizadas a la extensión

De esta manera fueron validadas las entradas de datos en las interfaces de usuario. Se demostró que la extensión da cumplimiento de forma correcta a todos los requisitos funcionales definidos, obteniéndose los resultados esperados. Además se comprobó la integración satisfactoria de la extensión EstimaPostArq a la herramienta CASE “Visual Paradigm for UML”.

Conclusiones parciales

Durante la realización de este capítulo se realizó el flujo de trabajo de implementación y pruebas. Dentro de la implementación se describieron los elementos del diseño en términos de 14 componentes y sus conexiones, facilitando el proceso de desarrollo de la extensión. Se definieron los estándares de codificación necesarios para el entendimiento, mantenimiento y por ende un mayor rendimiento del sistema, tomando como base el estándar de codificación del lenguaje Java. Se realizaron pruebas de desarrollador, específicamente pruebas funcionales a través del método de Caja Negra, aplicando la técnica de partición de equivalencia. Quedaron definidos los diseños de casos de pruebas que permitieron que se demostrara el cumplimiento de la extensión con los requisitos funcionales definidos. Se realizaron tres iteraciones de pruebas detectándose ocho anomalías (no conformidades), que fueron resueltas progresivamente validando la correcta implementación de la extensión.

CONCLUSIONES GENERALES

Como resultado del presente trabajo de diploma y dando cumplimiento a los objetivos propuestos, se arribaron a las siguientes conclusiones:

- A partir del estudio de las diferentes herramientas, tecnologías y metodologías de desarrollo de software se definieron las más idóneas para ser utilizadas o aplicadas en el desarrollo de la extensión. Su puesta en práctica permitió que la estructura, composición y desarrollo de la extensión se realizara de manera más efectiva y fácil. El estudio de los métodos de estimación más utilizados, haciendo énfasis en el Método de Estimación Post-Arquitectura UCI, posibilitó la identificación y posterior implementación de los requisitos funcionales y no funcionales de la extensión.
- El análisis y diseño del negocio permitió una mayor organización de las características y funcionalidades de la extensión. Gracias al diagrama de casos de uso quedaron recogidos en solo dos casos de uso los diez requisitos funcionales identificados, favoreciendo el desarrollo de la misma. Por otra parte, el diagrama de clases del diseño facilitó el entendimiento de la distribución de las clases y con el diagrama de secuencia se hizo más gráfico el flujo de información entre cada una de estas.
- La aplicación de patrones arquitectónicos como el Modelo-Vista-Controlador y de patrones de diseño garantizó la obtención de una arquitectura robusta y una correcta asignación de responsabilidades a cada una de las clases implementadas. La validación de las funcionalidades implementadas a través de pruebas funcionales utilizando el método de Caja Negra hizo posible la detección de errores en el comportamiento de la extensión. Esto garantizó la correcta implementación de cada uno de los requisitos funcionales de la extensión definidos.

RECOMENDACIONES

Realizar una extensión que aplique otros métodos de estimación conocidos, ejemplo el Método de Estimación Por Puntos de Casos de Uso, que permita contribuir al desarrollo de proyectos y facilitar el proceso de estimación.

Referencias Bibliográficas

REFERENCIAS BIBLIOGRÁFICAS

1. Enciclopedia Cubana en la Red. *Gestión de Proyecto.* [En línea] marzo de 2011. http://www.ecured.cu/index.php/Gesti%C3%B3n_de_proyecto.

2. Enciclopedia Cubana en la Red. *Planificación y Control en la Gestión de Proyectos.* [En línea] abril de 2010. http://www.ecured.cu/index.php/Planificaci%C3%B3n_y_Control_en_la_Gesti%C3%B3n_de_Proyectos.

3. Enciclipedia Cubana en la Red. *Estimación de tiempo y esfuerzo en proyectos de software.* [En línea] enero de 2012. http://www.ecured.cu/index.php/Estimaci%C3%B3n_de_tiempo_de_esfuerzo_en_proyectos_de_software.

4. **Rafael Dacal Diaz, Jorge G Tamayo.** "Herramienta para la estimación de proyecto". *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] 2010. [Citado el: 15 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_03072_10.

5. **Daimaris Céspedes Saíz, Grethel Martínez Rodríguez.** Propuesta de un método para la estimación del Modelado de Negocio y la Ingeniería de Requisitos. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] 2010. http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_03142_10.

6. CALISOFT. *Centro Nacional de Calidad de Software.* [En línea] http://calisoft.uci.cu/.

7. **Alejandro Hereaux Limonta.** Modelo de Referencia para el proceso de Planificación en los proyectos del Polo Gestión Biomédica. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] 2008. [Citado el: 23 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_1795_08.

8. **Flor Ramírez.** Principios de la Planeación. [En línea] 2008. [Citado el: 5 de Diciembre de 2012.] http://www.elprisma.com/apuntes/administracion_de_empresas/planeacion/..

9. **Fernando Gómez Fernández.** [En línea] 2008. http://oa.upm.es/1105/1/PFC_FERNANDO_GOMEZ_FERNANDEZ.pdf.

10. **Mayelin Timoteo Guerra.** “Plugin de Visual Paradigm for UML para la aplicación del Método de Estimación UCI.”. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] julio de 2011. [Citado el: 23 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05811_12.

11. Slideshare. *Modelo COCOMO.* [En línea] 2011. http://www.slideshare.net/animrak/modelo-cocomo.

12. **Adriana Gómez, María del C.López,Silvina Migani, Alejandra Otazú,.** COCOMO, UN MODELO DE ESTIMACION DE PROYECTOS DE SOFTWARE. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] [Citado el: 5 de Diciembre de 2012.] http://biblioteca.uci.cu/.

13. Slideshare. *COCOMO II.* [En línea] 2011. http://www.slideshare.net/equipo2/cocomo-ii.

14. Asignaturas. *Clase4.ppt.* [En línea] 2008. www.ing.unp.edu.ar/asignaturas/is/clases/clase4.ppt.

15. Kybeleconsulting. *Estimación por Puntos de Caso de Uso.* [En línea] 2008. http://www.kybeleconsulting.com/articulos/estimacion-puntos-caso-de-uso/.

16. Un poco de Java. *Estimación por Puntos de Caso de Uso (el famoso Karner).* [En línea] 2008. http://unpocodejava.wordpress.com/2010/11/06/estimacion-por-puntos-de-caso-de-uso-el-famoso-karner/.

17. **Jenny Suárez Vázquez.** *Programa de mejoras. 5414_Bases del Método de Estimación Post Arquitectura UCI.* [pdf] Ciudad de La Habana : s.n., 2011.

18. Enciclopedia Cubana en la Red. *UML.* [En línea] 2008. http://www.ecured.cu/index.php/UML.

Referencias Bibliográficas

19. Visual Paradigm for UML. *Visual Paradigm for UML 8.0 Released.* [En línea] 16 de Agosto de 2010. [Citado el: 1 de diciembre de 2012.] http://www.visual-paradigm.com/.
20. Slideshare. *Metodología OpenUp.* [En línea] 2006. http://www.slideshare.net/samith/metodologia-upen-up-3439131.
21. Enciclopedia Cubana en la Red. *OpenUp.* [En línea] 2009. http://www.ecured.cu/index.php/OpenUP.
22. Enciclopedia Cubana en la Red. *Java.* [En línea] 2010. http://www.ecured.cu/index.php/Java.
23. Java. *What is java.* [En línea] 2008. http://www.java.com/es/download/whatis_java.jsp.
24. Enciclopedia Cubana en la Red. *Visual Paradigm.* [En línea] 2008. http://www.ecured.cu/index.php/Visual_Paradigm.
25. NetBeans IDE. *NetBeans IDE .* [En línea] 17 de Octubre de 2011. [Citado el: 1 de diciembre de 2012.] http://netbeans.org.
26. Softonic. *NetBeans IDE.* [En línea] septiembre de 2012. [Citado el: 1 de Diciembre de 2012.] http://netbeans-ide.softonic.com/.
27. Enciclopedia Cubana en la Red. *Modelo de Dominio.* [En línea] 2011. http://www.ecured.cu/index.php/Modelo_de_dominio.
28. Enciclopedia Cubana en la Red. *Gestión de Requisitos.* [En línea] enero de 2009. http://www.ecured.cu/index.php/Gesti%C3%B3n_de_Requisitos.
29. Enciclopedia Cubana en la Red. *Flujo de Trabajo Requisitos: Actores del Sistema.* [En línea] 2009. http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento#Actores_del_Sistema.
30. Enciclopedia Cubana en la Red. *Flujo de Trabajo Requisitos: Diagrama de Casos de Uso.* [En línea] 2009.
http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento#Diagrama_de_Casos_de_Uso_del_Sistema.
31. Enciclopedia Cubana en la Red. *Flujo de Trabajo Analisis y Diseño.* [En línea] 2010. http://www.ecured.cu/index.php/Flujo_de_Trabajo_An%C3%A1lisis_y_Dise%C3%B1o.
32. Slideshare. *Diagrama de secuencia.* [En línea] 20011. www.slideshare.net/.../diagramas-de-secuencia-presentation.
33. Enciclopedia Cubana en la Red. *Patrones de Diseño y Arquitectura.* [En línea] 2011. http://www.ecured.cu/index.php/Patrones_de_dise%C3%B1o_y_arquitectura.
34. Enciclopedia Cubana en la Red. *Patrones Arquitectónicos.* [En línea] 2011. http://www.ecured.cu/index.php/Patrones_arquitect%C3%B3nicos.
35. Enciclopedia Cubana en la Red. *Factory Method.* [En línea] 2011. http://www.ecured.cu/index.php/Factory_Method.
36. Microsoft. *Singleton.* [En línea] 2006. msdn.microsoft.com/es-es/library/bb972272.aspx.
37. Enciclopedia Cubana en la Red. *Iterator.* [En línea] 2010. http://www.ecured.cu/index.php/Iterator.
38. Enciclopedia Cubana en la Red. *Patrones de Asignación de Responsabilidades.* [En línea] 2010. http://www.ecured.cu/index.php/Patrones_de_Asignaci%C3%B3n_de_Responsabilidades.
39. Enciclopedia Cubana en la Red. *Diagrama de despliegue.* [En línea] 2008. http://www.ecured.cu/index.php/Diagrama_de_despliegue.

40. Enciclopedia Cubana en la Red. *Flujo de Trabajo Implementación.* [En línea] 2010. http://www.ecured.cu/index.php/Flujo_de_Trabajo_de_Implementaci%C3%B3n.
41. Slideshare. *Diagrama de componentes.* [En línea] 2008. www.slideshare.net/dhuertacruz/diagrama-de-componentes-15705604.
42. **Hommel, Scott.** Convenciones de Código para el lenguaje de programación JAVA™. [En línea] 20 de abril de 1999. http://www.javahispano.com.
43. Enciclopedia Cubana en la Red. *Pruebas de Software.* [En línea] 2011. http://www.ecured.cu/index.php/Pruebas_de_software.
44. Enciclopedia Cubana en la Red. *Pruebas de Caja Negra.* [En línea] 2010. http://www.ecured.cu/index.php/Pruebas_de_caja_negra.
45. Enciclopedia Cubana en la Red. *Estrategia de pruebas de software.* [En línea] 2010. http://www.ecured.cu/index.php/Estrategia_de_pruebas_de_software.

Bibliografías

BIBLIOGRAFIA

1. Enciclopedia Cubana en la Red. *Gestión de Proyecto.* [En línea] marzo de 2011. http://www.ecured.cu/index.php/Gesti%C3%B3n_de_proyecto.

2. Enciclopedia Cubana en la Red. *Planificación y Control en la Gestión de Proyectos.* [En línea] abril de 2010. http://www.ecured.cu/index.php/Planificaci%C3%B3n_y_Control_en_la_Gesti%C3%B3n_de_Proyectos.

3. Enciclipedia Cubana en la Red. *Estimación de tiempo y esfuerzo en proyectos de software.* [En línea] enero de 2012. http://www.ecured.cu/index.php/Estimaci%C3%B3n_de_tiempo_de_esfuerzo_en_proyectos_de_software.

4. **Rafael Dacal Diaz, Jorge G Tamayo.** "Herramienta para la estimación de proyecto". *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] 2010. [Citado el: 15 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_03072_10.

5. **Daimaris Céspedes Saíz, Grethel Martínez Rodríguez.** Propuesta de un método para la estimación del Modelado de Negocio y la Ingeniería de Requisitos. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] 2010. http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_03142_10.

6. CALISOFT. *Centro Nacional de Calidad de Software.* [En línea] http://calisoft.uci.cu/.

7. **Alejandro Hereaux Limonta.** Modelo de Referencia para el proceso de Planificación en los proyectos del Polo Gestión Biomédica. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] 2008. [Citado el: 23 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_1795_08.

8. **Flor Ramírez.** Principios de la Planeación. [En línea] 2008. [Citado el: 5 de Diciembre de 2012.] http://www.elprisma.com/apuntes/administracion_de_empresas/planeacion/..

9. **Fernando Gómez Fernández.** [En línea] 2008. http://oa.upm.es/1105/1/PFC_FERNANDO_GOMEZ_FERNANDEZ.pdf.

10. **Mayelin Timoteo Guerra.** “Plugin de Visual Paradigm for UML para la aplicación del Método de Estimación UCI.”. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] julio de 2011. [Citado el: 23 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05811_12.

11. Slideshare. *Modelo COCOMO.* [En línea] 2011. http://www.slideshare.net/animrak/modelo-cocomo.

12. **Adriana Gómez, María del C.López,Silvina Migani, Alejandra Otazú,.** COCOMO, UN MODELO DE ESTIMACION DE PROYECTOS DE SOFTWARE. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] [Citado el: 5 de Diciembre de 2012.] http://biblioteca.uci.cu/.

13. Slideshare. *COCOMO II.* [En línea] 2011. http://www.slideshare.net/equipo2/cocomo-ii.

14. Asignaturas. *Clase4.ppt.* [En línea] 2008. www.ing.unp.edu.ar/asignaturas/is/clases/clase4.ppt.

15. Kybeleconsulting. *Estimación por Puntos de Caso de Uso.* [En línea] 2008. http://www.kybeleconsulting.com/articulos/estimacion-puntos-caso-de-uso/.

16. Un poco de Java. *Estimación por Puntos de Caso de Uso (el famoso Karner).* [En línea] 2008. http://unpocodejava.wordpress.com/2010/11/06/estimacion-por-puntos-de-caso-de-uso-el-famoso-karner/.

17. **Jenny Suárez Vázquez.** *Programa de mejoras. 5414_Bases del Método de Estimación Post Arquitectura UCI.* [pdf] Ciudad de La Habana : s.n., 2011.

18. Enciclopedia Cubana en la Red. *UML.* [En línea] 2008. http://www.ecured.cu/index.php/UML.

19. Visual Paradigm for UML. *Visual Paradigm for UML 8.0 Released.* [En línea] 16 de Agosto de 2010. [Citado el: 1 de diciembre de 2012.] http://www.visual-paradigm.com/.
20. Slideshare. *Metodología OpenUp.* [En línea] 2006. http://www.slideshare.net/samith/metodologia-upen-up-3439131.
21. Enciclopedia Cubana en la Red. *OpenUp.* [En línea] 2009. http://www.ecured.cu/index.php/OpenUP.
22. Enciclopedia Cubana en la Red. *Java.* [En línea] 2010. http://www.ecured.cu/index.php/Java.
23. Java. *What is java.* [En línea] 2008. http://www.java.com/es/download/whatis_java.jsp.
24. Enciclopedia Cubana en la Red. *Visual Paradigm.* [En línea] 2008. http://www.ecured.cu/index.php/Visual_Paradigm.
25. NetBeans IDE. *NetBeans IDE .* [En línea] 17 de Octubre de 2011. [Citado el: 1 de diciembre de 2012.] http://netbeans.org.
26. Softonic. *NetBeans IDE.* [En línea] septiembre de 2012. [Citado el: 1 de Diciembre de 2012.] http://netbeans-ide.softonic.com/.
27. Enciclopedia Cubana en la Red. *Modelo de Dominio.* [En línea] 2011. http://www.ecured.cu/index.php/Modelo_de_dominio.
28. Enciclopedia Cubana en la Red. *Gestión de Requisitos.* [En línea] enero de 2009. http://www.ecured.cu/index.php/Gesti%C3%B3n_de_Requisitos.
29. Enciclopedia Cubana en la Red. *Flujo de Trabajo Requisitos: Actores del Sistema.* [En línea] 2009. http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento#Actores_del_Sistema.
30. Enciclopedia Cubana en la Red. *Flujo de Trabajo Requisitos: Diagrama de Casos de Uso.* [En línea] 2009.
http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento#Diagrama_de_Casos_de_Uso_del_Sistema.
31. Enciclopedia Cubana en la Red. *Flujo de Trabajo Analisis y Diseño.* [En línea] 2010. http://www.ecured.cu/index.php/Flujo_de_Trabajo_An%C3%A1lisis_y_Dise%C3%B1o.
32. Slideshare. *Diagrama de secuencia.* [En línea] 20011. www.slideshare.net/.../diagramas-de-secuencia-presentation.
33. Enciclopedia Cubana en la Red. *Patrones de Diseño y Arquitectura.* [En línea] 2011. http://www.ecured.cu/index.php/Patrones_de_dise%C3%B1o_y_arquitectura.
34. Enciclopedia Cubana en la Red. *Patrones Arquitectónicos.* [En línea] 2011. http://www.ecured.cu/index.php/Patrones_arquitect%C3%B3nicos.
35. Enciclopedia Cubana en la Red. *Factory Method.* [En línea] 2011. http://www.ecured.cu/index.php/Factory_Method.
36. Microsoft. *Singleton.* [En línea] 2006. msdn.microsoft.com/es-es/library/bb972272.aspx.
37. Enciclopedia Cubana en la Red. *Iterator.* [En línea] 2010. http://www.ecured.cu/index.php/Iterator.
38. Enciclopedia Cubana en la Red. *Patrones de Asignación de Responsabilidades.* [En línea] 2010. http://www.ecured.cu/index.php/Patrones_de_Asignaci%C3%B3n_de_Responsabilidades.
39. Enciclopedia Cubana en la Red. *Diagrama de despliegue.* [En línea] 2008. http://www.ecured.cu/index.php/Diagrama_de_despliegue.

40. Enciclopedia Cubana en la Red. *Flujo de Trabajo Implementación.* [En línea] 2010. http://www.ecured.cu/index.php/Flujo_de_Trabajo_de_Implementaci%C3%B3n.
41. Slideshare. *Diagrama de componentes.* [En línea] 2008. www.slideshare.net/dhuertacruz/diagrama-de-componentes-15705604.
42. **Hommel, Scott.** Convenciones de Código para el lenguaje de programación JAVA™. [En línea] 20 de abril de 1999. http://www.javahispano.com.
43. Enciclopedia Cubana en la Red. *Pruebas de Software.* [En línea] 2011. http://www.ecured.cu/index.php/Pruebas_de_software.
44. Enciclopedia Cubana en la Red. *Pruebas de Caja Negra.* [En línea] 2010. http://www.ecured.cu/index.php/Pruebas_de_caja_negra.
45. Enciclopedia Cubana en la Red. *Estrategia de pruebas de software.* [En línea] 2010. http://www.ecured.cu/index.php/Estrategia_de_pruebas_de_software.
46. **Pressman, Roger S.** *"Ingeniería del software: Un enfoque práctico. Quinta edición".* McGraw- Hill : s.n., 2002.
47. **López, C. Romero.** *Técnicas de Programación y Control de Proyectos.* s.l. : Ediciones Pirámide, 1997.
48. Enciclopedia Cubana en la Red. *Metodologías de desarrollo de software: OpenUP.* [En línea] 24 de septiembre de 2012. [Citado el: 27 de enero de 2013.] http://www.ecured.cu/index.php/OpenUp.
49. OpenUP como alternativa metodológica para proyectos pequeños de software. *Enciclopedia Cubana en la Red.* [En línea] 27 de Septiembre de 2008. [Citado el: 24 de noviembre de 2012.] http://www.ecured.cu/index.php/Desarrollo_%C3%A1gil_de_software.
50. Microsoft. *Estándares de Codificación.* [En línea] 2008. msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx.
51. **Aliemny Avalos Abrahantes, Yenma Macias Gil,.** Propuesta de proceso para la estimación en entidades de servicios TI (Tecnología de la Información) en la UCI. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] Junio de 2011. [Citado el: 15 de Noviembre de 2012.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_04240_11.
52. **Lianet Cabrera González y Enrique Roberto Pompa Torres.** Extensión de Visual Paradigm for UML para el desarrollo dirigido por modelos de aplicaciones de gestión de información. *Serie Científica de la Universidad de las Ciencias Informáticas.* [En línea] 15 de Octubre de 2012. [Citado el: 23 de Noviembre de 2012.] http://publicaciones.uci.cu/index.php/SC.
53. Enciclopedia Cubana en la Red. *Modelo de Dominio.* [En línea] enero de 2012. [Citado el: 14 de febrero de 2013.] http://www.ecured.cu/index.php/Modelo_de_dominio.
54. Enciclopedia Cubana en la Red. *Metodologías para el diseño de aplicaciones multimedia.* [En línea] noviembre de 2012. http://www.ecured.cu/index.php/Metodolog%C3%ADas_para_el_dise%C3%B1o_de_aplicaciones_multimedia.
55. Enciclopedia Cubana en la Red. *Metodología Ágil.* [En línea] enero de 2012. http://www.ecured.cu/index.php/Metodolog%C3%ADa_%C3%A1gil.
56. Enciclopedia Cubana en la Red. *EcuRed.* [En línea] enero de 2012. [Citado el: 14 de febrero de 2013.] htpp://www.ecured.cu/.

57. Enciclopedia Cubana en la Red. *Planeación de proyectos.* [En línea] marzo de 2011. http://www.ecured.cu/index.php/Planeaci%C3%B3n_de_proyecto.
58. Enciclopedia Cubana en la Red. *Gestión de Proyectos de Software.* [En línea] junio de 2011. http://www.ecured.cu/index.php/Gesti%C3%B3n_de_Proyectos_de_Software.
59. Enciclopedia Cubana en la Red. *IDE de Programación.* [En línea] 2010. http://www.ecured.cu/index.php/IDE_de_Programaci%C3%B3n.
60. Enciclopedia Cubana en la Red. *NetBeans.* [En línea] 2010. http://www.ecured.cu/index.php/NetBeans.
61. Enciclopedia Cubana en la Red. *Planeación de proyecto: Estimación de esfuerzo y costo.* [En línea] junio de 2010. http://www.ecured.cu/index.php/Planeaci%C3%B3n_de_proyecto#Estimaci.C3.B3n_el_esfuerzo_y_costo.
62. Enciclopedia Cubana en la Red. *Lenguaje Unificado de Modelado.* [En línea] http://www.ecured.cu/index.php/Lenguaje_de_Modelaje_Unificado.
63. **GT, Susana.** EcuRed. *Enciclopedia Cubana en la Red.* [En línea] 28 de Septiembre de 2012. [Citado el: 1 de Diciembre de 2012.] http://www.ecured.cu.
64. **Cerillo, David.** *Estimación del software.* [En línea] 1999. [Citado el: 5 de Diciembre de 2012.] http://alarcos.inf-cr.uclm.es/DOC/PGSI/doc/esp/T9899_DCerrillo.pdf...

GLOSARIO

Código fuente: Es la escritura original, en un lenguaje de programación determinado, de un programa específico, o sea, los comandos o instrucciones escritas para el mismo.

Estimación: Determinación del valor de uno o más parámetros estadísticos a partir de las observaciones de una muestra.

Extensible: Que se puede extender, que puede ser extendido.

Extensión o Plugin: Es una noción que no forma parte del diccionario de la Real Academia Española (RAE). Se trata de un concepto de la lengua inglesa que puede entenderse como “inserción” y que se emplea en el campo de la informática. Un plugin es aquella aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. En nuestro idioma, por lo tanto, puede nombrarse al plugin como un complemento o extensión.

Herramienta: Del latín *ferramenta*. Instrumento que permite realizar ciertos trabajos. Estos objetos fueron diseñados para facilitar la realización de una tarea mecánica que requiere del uso de una cierta fuerza. El concepto de herramienta también se utiliza para nombrar a cualquier procedimiento que mejora la capacidad de realizar ciertas tareas. De esta forma, es posible hablar de herramientas informáticas: “Microsoft Office es una herramienta para desarrollar tareas de oficina”.

Iteración: Del latín *iteratio*. Describe el acto y consecuencia de iterar. Verbo que se emplea como sinónimo de reiterar o repetir (entendidos como volver a desarrollar una acción o pronunciar de nuevo lo que ya se había dicho). Para la programación, la iteración consiste en reiterar un conjunto de instrucciones o acciones con uno o varios objetivos.

Métrica: Del latín *metrĭcus*. Se define como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Es la medida que permite reconocer a un sistema de información o software.

Portabilidad: Cualidad de un programa, de un sistema operativo o de un lenguaje que se puede ejecutar en diversos tipos de ordenador.

Regresión: Retroceso o vuelta hacia atrás de un proceso.

Usabilidad: El concepto proviene del inglés *usability.* Hace referencia a la facilidad con que un usuario puede utilizar una herramienta fabricada por otras personas, con el fin de alcanzar un cierto objetivo.