Universidad de las Ciencias Informáticas

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

Título: Arquitectura del Módulo de Modelado para la Suite de IGSW.

Autor: Elizabeth I. Peñate del Rio.

Tutor: Ing. Yusleydi Fernández del Monte.

Ing. Lisandra Cala Hernández.

Ing. Mairim Delgado Muñiz

La Habana, Cuba, Junio 2013

"Año 55 de la Revolución"

“Aplica tu corazón a la enseñanza,
Y tus oídos a las palabras de sabiduría.”

Salomón.

Declaración de Autoría

Se declara que la única autora de este trabajo es Elizabeth Idalia Peñate del Rio y se autoriza a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste se firma a los ____ días del mes de ____ del año ____ el presente documento.

_____
Elizabeth Idalia Peñate del Rio.

_____ _____
Ing. Yusleydi Fernández del Monte. Ing. Mairim Delgado Muñiz.

Dedico este trabajo de diploma a mi abuelita Idalia Bolmey Hernández y a mi madre Meidis del Rio Bolmey. Por ser ese motor que me impulsó a venir a la universidad y estar hasta el final, por la dedicación con la que cada día oraban por mí, para que mi Padre Celestial me sostuviera día tras día. Por brindarme su amor y apoyo incondicional en cada etapa de mi vida, y lograr así la persona que soy, gracias. Por tanto todo el esfuerzo realizado en este trabajo de diploma lo dedico a ellas. Dios les bendiga abuelita y mami.

No puedo concluir esta etapa sin agradecer a las personas que han estado a mi lado brindándome apoyo de manera incondicional. Entre ellos se encuentran mis amigos: Dailencita (florecita) que me brindó palabras de aliento en los momentos que más lo necesitaba y Carlos (el negro) por hacerme reír en los momentos más difíciles de esta etapa.

A mí querido hermano Adrián y a Orlando por estar a mi lado y socorrerme cada vez que los necesitaba.

Los hermanos: José Antonio, Fidel, Alexey Alayo y Annier por brindarme su apoyo con los diagramas.

A mis compañeros de proyecto: Iliana, Carlos Julio y Ana Evis.

A Samuel por todos los dolores de cabeza que le di.

Además agradecer a las tutoras: Lisandra Cala, Yusleidi del Monte y Mairin Delgado.

A mis padres que con su devoción y cuidado se han mantenido pendientes de mí y me han dado su apoyo moral.

Y por último pero no menos importante a mi Señor Jesucristo, por su salvación y guía durante estos 5 años de carrera. Toda la Gloria sea a Él.

Resumen

Actualmente no existe una herramienta libre de modelado de software que sea recomendable y brinde todas las funcionalidades que necesite el usuario, provocando que el modelado de software no sea el mejor. El proyecto Nova-QALIT se encuentra inmerso en el desarrollo de una Suite de Ingeniería de Software (ISW). Dicha Suite no cuenta con un módulo de modelado de software, por lo que el objetivo principal de la presente investigación es determinar una propuesta de la arquitectura de software sobre la que se desarrolla el módulo de modelado de la Suite de ISW que permita la representación de modelos de software utilizando las notaciones Lenguaje de Modelado Unificado (UML) y Notación para el Modelado de Procesos de Negocio (BPMN). Durante la investigación se analizaron los elementos fundamentales de la arquitectura de software, con el objetivo de determinar una arquitectura robusta que permita la representación de modelos de software utilizando estas notaciones. Además se evalúa la arquitectura haciendo uso del método de evaluación Análisis de Desventajas de Arquitectura (ATAM) y de los atributos de calidad de la norma ISO 9126 determinándose solo tres riesgos que se mitigaron con diferentes decisiones arquitectónicas tomadas, lo cual representa un resultado satisfactorio.

Palabras Claves: Arquitectura de software, BPMN, Suite de ISW, UML.

ÍNDICE

Índice de figuras

Índice de tablas

Introducción

El desarrollo de software ha ascendido en la última década. Es un proceso complejo y difícil, por lo que se han creado diferentes herramientas que permiten simplificar y lograr mayor calidad en el mismo. Un componente importante en entornos de desarrollo de software son las herramientas de modelado ya que permiten el enfoque y la atención a los aspectos relevantes del sistema mediante el diseño de modelos. De esta manera se logra representar lo que se desea desarrollar, se minimiza el costo, los riesgos y los cambios de los requerimientos del usuario.

Un modelo de software es una abstracción que simplifica la realidad, suprimiendo detalles irrelevantes y reteniendo los aspectos esenciales, de modo que lo principal del sistema sea mejor conocido y pueda hacer frente a su complejidad [1]. El proceso de modelado es útil tanto en pequeños como en grandes sistemas. A través del mismo, se enfatiza en ciertas propiedades críticas del sistema, ayudando al ingeniero de software a visualizar la aplicación a construir.

A nivel mundial existen varias herramientas de modelado libres y privativas. En su mayoría las herramientas libres presentan desventajas que atentan contra la calidad y el desarrollo de una aplicación informática. Solo brindan soporte para la notación UML y obvian notaciones que van enfocadas al modelo de negocio y las bases de datos. No tienen la madurez y las funcionalidades necesarias para obtener un resultado de la mejor calidad.

Las herramientas privativas son las predominantes y más utilizadas, le brindan al usuario las funcionalidades que necesita para hacer su trabajo, optimizando la comunicación entre clientes y desarrolladores. Propician un conjunto de ayuda para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Resaltar que al ser propietaria, se debe pagar la licencia de uso, siendo para Cuba una desventaja.

Por tanto, el país en el desarrollo de software, ha tenido la necesidad de utilizar las opciones que brindan las herramientas de modelado privativas. El departamento de Sistemas Operativos (SO) del Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas (UCI) no está exento de esto, sin dejar atrás la migración que está llevando a cabo el país.

Teniendo en cuenta que el uso de herramientas privativas representa una desventaja para Cuba y la ausencia de una herramienta libre de modelado de software que sea suficiente, el proyecto Nova QALIT del departamento SO se encuentra inmerso en la realización de una Suite de Ingeniería de Software (IGSW), que responda con los principios de software libre y brinde las funcionalidades que necesita el

usuario. La misma no cuenta con un módulo de modelado de software que permita la representación arquitectónica de software.

Por lo antes expuesto se identifica como **problema científico**: ¿Cómo personalizar la representación de modelos de software utilizando las notaciones UML y BPMN?

Se define como **objeto de estudio** la arquitectura de software, seleccionándose como **campo de acción** la arquitectura para herramientas de modelado de software.

Para el desarrollo de esta investigación se plantea como **idea a defender**:
La arquitectura para el módulo de modelado de la suite de Ingeniería de Software puede permitir la representación de modelos de software utilizando las notaciones UML y BPMN.

El **objetivo general** de esta investigación es diseñar la arquitectura de una herramienta que permita la representación de modelos de software utilizando las notaciones UML y BPMN.

Como **objetivos específicos** se plantearon los siguientes:

1. Caracterizar las herramientas de modelado de software.
2. Definir las diferentes vistas arquitectónicas de la herramienta de modelado que garantice la representación de los modelos que forman parte de la arquitectura de software.
3. Evaluar la arquitectura propuesta para la herramienta de modelado de software.

Para dar cumplimiento a los objetivos específicos se formularon las siguientes **tareas de investigación**:

1. Caracterización de las herramientas de modelado de software existentes para determinar los aspectos fundamentales a tener en cuenta.
2. Selección de la arquitectura de la herramienta CASE para el módulo de modelado de software.
3. Levantamiento de requerimientos funcionales y no funcionales del módulo de modelado para caracterizar el producto.
4. Diseño del módulo de modelado de la Suite de IGSW para definir las diferentes vistas arquitectónicas.
5. Evaluación de la arquitectura de la herramienta CASE desarrollada para verificar si puede ser capaz de resolver el problema planteado.

Para el desarrollo de esta investigación se utilizaron los siguientes **métodos científicos:**

Métodos Teóricos:

Analítico-Sintético: Permite realizar un análisis del desarrollo y funcionamiento de la arquitectura de herramientas de modelado, para así sintetizar en sus principales aspectos, funcionalidades y características.

Análisis Histórico Lógico: Permite hacer un análisis completo de la evolución de las diferentes arquitectura de las herramientas de modelado existentes, seleccionando las más utilizadas, para así obtener las características e información necesaria para el diseño de la arquitectura de la herramienta de modelado de software para entornos libres.

Métodos Empíricos:

Experimental: Al terminar la arquitectura de la herramienta, se realizan las pruebas para verificar si es correcta.

El presente trabajo de diploma se encuentra estructurado en tres capítulos:

Capítulo#1: Fundamentación teórica de las herramientas CASE para el modelado de software.

En este capítulo se exponen los resultados obtenidos al investigar sobre la arquitectura de herramientas de modelado de software y sus características esenciales.

Capítulo#2: Obtención de requisitos y diseño de la arquitectura de la herramienta de modelado para la Suite de Ingeniería de Software.

En este capítulo se realiza el diseño de la arquitectura del sistema y se describen los procesos que se emplearon para el diseño del módulo de modelado.

Capítulo#3: Evaluación de la arquitectura de la herramienta de modelado para la Suite de Ingeniería de Software.

En este capítulo se presentan los resultados de evaluar la arquitectura de la herramienta de modelado para determinar el cumplimiento de los requisitos.

Capítulo #1: Fundamentación teórica de las herramientas CASE para el modelado de software

Introducción

En este capítulo se exponen los resultados obtenidos de investigar sobre la arquitectura de herramientas de modelado de software y sus características esenciales. Se abordan una serie de elementos que son fundamentales para la comprensión del diseño que se propone. Se definen algunos conceptos y características que sirven de apoyo para el conocimiento del tema. Asimismo, se plasma la metodología de desarrollo y se hace referencia al lenguaje de programación a utilizarse, además de otros aspectos de interés.

1. Definición de Herramienta CASE

Herramienta Computer Aided Software Engineering (CASE), Ingeniería de Software Asistida por Computadora.

1.1 Concepto de Herramientas CASE

José M. de las Heras define herramienta CASE como la creación de software utilizando técnicas de diseño y metodologías de desarrollo bien definidas, soportadas por herramientas automatizadas operativas en el ordenador [2].

Una herramienta CASE es además un producto computacional enfocado a apoyar una o más técnicas dentro de un método de desarrollo de software [3].

Principalmente nacen para solucionar el problema de la mejora de la calidad del desarrollo de sistemas de mediano y gran tamaño, y en segundo término por el aumento de la productividad. [4]

En el marco de este trabajo se asume como concepto de herramientas CASE:

Conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases. [5]

1.2 Características de Herramientas CASE

Las herramientas CASE proveen beneficios en todas las etapas del proceso de desarrollo de software, realizan el trabajo de diseño de software más fácil y agradable. [4]

Características básicas:

Permiten a los usuarios dibujar diagramas para la planificación, análisis, o diseño en la pantalla de un ordenador.

- ✓ Solicitan información acerca de cada uno de los objetos de un diagrama y de las interrelaciones de dichos objetos.
- ✓ Almacenan el significado del diagrama, en vez del diagrama en sí mismo, dentro de un depósito de información (repositorio).

- ✓ Comprueba la exactitud, integridad y completitud de cada diagrama. Los diagramas que se ofrezcan deben ser elegidos con el objetivo de facilitar esta labor.[6]

1.3 Importancia de Herramientas CASE

El uso de herramientas CASE permite una mejora de la calidad de los desarrollos realizados, logrando el aumento de la producción a través de la automatización de determinadas tareas. [5]

La tecnología CASE trae consigo beneficios al desarrollo del software como:

1. Facilidad para la revisión de aplicaciones: En varias ocasiones una vez que se implementa una aplicación, se emplea por mucho tiempo. Cuando se crea un sistema en grupo, el contar con un almacenamiento central de la información agiliza el proceso de revisión ya que este proporciona consistencia y control de estándares. La capacidad que brindan algunas herramientas para la generación de códigos contribuye a modificar el sistema por medio de las especificaciones en los diagramas más que por cambios directamente al código fuente.
2. Ambiente interactivo en el proceso de desarrollo: El desarrollo del sistema es un proceso interactivo. Las herramientas CASE soportan pasos interactivos al eliminar diagramas manualmente. Como resultado de esto, los analistas pueden repasar y revisar los detalles del sistema con mayor frecuencia y de forma segura.
3. Generación de código: La generación de código trae consigo beneficios considerables dentro del desarrollo del software, permite acelerar el proceso de desarrollo al automatizar el tedioso proceso de crear diagramas y posteriormente codificarlos. Asegura una estructura consistente (lo que ayuda en la fase de mantenimiento) y disminuye la ocurrencia de varios tipos de errores, mejorando de esta manera la calidad del software desarrollado. Además permite la reutilización, ya que se puede volver a utilizar el software y las estructuras estándares para crear otros sistemas o modificar el actual, así como el cambio de una especificación modular, es decir, volver a generar el código y los enlaces con otros módulos.
4. Reducción de costos: La reducción de costos se ve reflejada, al utilizar una herramienta CASE para diseño de interfaces, logrando hacer cambios de interfaz con rapidez. Y usando herramientas con generación de código, el código fuente se puede volver a generar después de hacer las modificaciones requeridas rápidamente en los diagramas.
5. Incremento en la productividad: Se ve reflejado por la optimización de los recursos con que se cuenta para llevar a cabo un proceso. Al hacer uso de una herramienta CASE se administran los recursos humanos y tecnológicos, y al cambiar los conocimientos de los desarrolladores con las prestaciones que brindan estas herramientas es que se obtiene un incremento en la productividad en el desarrollo del software. [4]

2. Herramientas CASE de modelado de software.

Las herramientas de modelado de software tienen gran importancia, ayudan al ingeniero de software a visualizar el sistema a construir. Permiten realizar una abstracción del objeto real, dando una idea de lo que ocurre en la realidad usándose como base en el desarrollo, facilitando la comunicación entre los desarrolladores y los clientes, logrando así la creación de un producto de alta calidad.

2.1 Herramientas CASE más utilizadas.

Visual Paradigm

Visual Paradigm para UML (Lenguaje de Modelado Unificado) es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, implementación y pruebas. Ayuda a una rápida construcción de aplicaciones de calidad, a un menor coste. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y generar documentación. Esta herramienta permite que los desarrolladores de software realicen modelos, durante la creación y distribución del proceso de desarrollo de aplicaciones. Además provee el modelado de procesos de negocios, y de un generador de mapeo de objetos-relacionales para los lenguajes de programación Java.NET y PHP. [7]

Entre sus principales funcionalidades:

- Creación de modelos UML.
- Modelado de base de datos.
- Mapa de relación de objetos.
- Interoperabilidad: intercambio de diagramas UML y modelos con otras herramientas, usando representaciones industriales comunes.
- Integración IDE[1].
- Modelado de requerimientos.
- Modelo de procesos de negocios: visualización, improvisación y entendimiento de procesos con la herramienta BPMN.
- Colaboración en equipo: compatible con servidores de equipo como VP Teamwork Server[2], etc.
- Generador de código.
- Generador de documentación.
- Navegación intuitiva entre el código y el modelo.
- Generador de documentación y reportes UML, PDF[3], HTML/MS, Word.
- Demanda en tiempo real, modelo incremental de viaje redondo y sincronización de código fuente.
- Superior entorno de modelado visual.

[1] **IDE**: Entorno de Dasarrollo Integrado

[2] **VP Teamwork Server**: es un repositorio para almacenar los diagramas de diseño y los archivos complementarios.

[3] **PDF**: es un formato de almacenamiento de documentos digitales independiente de plataformas de software o hardware.

- Diagramas de diseño automático sofisticado.
- Análisis de texto y soporte de tarjeta CRC[4]. [7]

Tipos de Diagramas que realiza:

Diagramas en el Visual Paradimg	
Diagramas de modelado estructurales.	Diagramas de modelado de comportamiento.
Diagrama de paquetes.	Diagrama de actividades.
Diagrama de clase.	Diagrama de serie o secuencia.
Diagrama de despliegue.	Diagrama de interacción de sobrevista.
Diagrama de objetos.	Diagrama de máquina de estados.
Diagrama de estructura compuesto.	Diagrama de comunicación.
Diagrama de componentes.	Diagrama de tiempo.
Diagrama entidad-relación.	Diagrama de casos de uso. [7]
Diagrama ORM[5].	Diagrama básico.
Diagrama de requerimientos.	Diagrama de tarjetas CRC.
Diagrama de procesos de negocio.	Diagrama de conversación.
Diagrama de flujo de datos.	Diagrama EPC[6].
Diagrama de mapa de procesos.	Diagrama WSDL[7].
Diagrama de mapa mental.	Diagrama fact.
Diagrama análisis.	Diagrama matrix.

Tabla 1: Diagramas Visual Paradimg

Rational Rose

Es una herramienta para el Modelado Visual mediante UML de sistemas de software, manteniendo así la consistencia de los modelos del software y el chequeo de la sintaxis UML, especificando, analizando y diseñando el sistema antes de codificarlo. [8]

Funcionalidades:

- Browser: Permiten navegar por los elementos de las vistas de Rose, añadir, borrar, renombrar y mover elementos de los modelos. Ver asociaciones (son un elemento más), abrir las especificaciones de algún elemento y agrupar en paquetes los elementos de los modelos. Además de añadir y abrir diagramas.
- Toolbars: Permite acceder a comandos específicos para crear elementos del diagrama. Depende del diagrama que se está visualizando en Diagrama Window.

[4] **CRC**: Clase, Responsabilidades, Colaboradores. Es una herramienta principalmente de diseño.

[5] **ORM:** Mapero objeto-relacional

[6] **EPC:** Cadenas de procesos condicionados por eventos.

[7] **WSDL:** Permite describir la interfaz de un servicio web en un formato XML.

- Diagrama Window: Permite crear y visualizar diagramas UML. Cambios en el diagrama se reflejan en el Browser y viceversa (múltiples cambios)
- Documentación Window: Permite documentar elementos de los modelos. Añadir documentación a un elemento del modelo. Formar parte de la especificación del elemento.
- Log: Permite ver resultados de los comandos y acciones llevadas a cabo. [8]

Tipos de Diagramas que realiza:

Diagramas en Rational Rose.	
Diagrama de actividades.	Diagrama de paquetes.
Diagrama de secuencia.	Diagrama de caso de uso.
Diagrama de colaboración.	Diagrama de componentes.
Diagrama de despliegues.	Diagramas de estado.
Diagrama de clases.	Diagrama de interacción. [8]

Tabla 2: Diagramas Rational Rose.

Enterprise Architect:

Enterprise Architect es una herramienta gráfica multiusuario que permite crear un software robusto y mantenible. Es una herramienta que abarca integralmente el ciclo de vida de un software, cubriendo el desarrollo de software desde el levantamiento de los requisitos, a través de las etapas de análisis, modelos de diseño, prueba y finalmente el mantenimiento y re-uso. Enterprise Architect combina la potencia de las últimas especificaciones UML 2.1 con un alto rendimiento y una interfaz intuitiva. [9]

Funcionalidades:

- Interfaz de usuario intuitiva: Posee un amplio rango de barras de herramientas, ventanas acoplables, y estilos visuales. Guarda y restaura disposiciones de ventanas personalizadas. Modifica y personaliza las barras de herramientas, de menús y crea sus propios aceleradores.
- Para transformaciones MDA[8]: La arquitectura dirigida por modelos permite transformar elementos simples en objetivos complejos y transformaciones completas dirigidas por plantillas. Estas son de transformaciones fáciles de escribir y modificar. Además genera y sincroniza directamente su modelo específico de plataforma desde su modelo independiente de plataforma.
- Documentación flexible y comprensible: Posee un generador RTF[9] conducido por plantillas completas, estas soportan todas las características de los elementos del modelo y los datos extendidos (tales como Pruebas, Riesgos, Recursos, Cambios, etc.). Además soportan encabezados, pies de página, tablas de contenidos, imágenes embebidas, índices, títulos de páginas, tablas jerarquizadas y más. Posee opciones de salida flexibles con filtros y criterios de

[8] **MDA**: es una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos, para dar soporte a la ingeniería dirigida a modelos de los sistemas de software.

[9] **RTF** es un formato de archivo para el intercambio de documentos multiplataforma.

selección, además guarda las plantillas reportadas para una reutilización posterior y genera informes HTML detallados.

- Ingeniería de código directa e inversa: Posee un generador de código dirigido por plantillas completas y agrega lenguajes de destino adicionales. Posee un editor del código fuente con capacidad para guardar y sincronizar con rapidez. [9]
- Soporte para UML 2.0.
- Generación de código e ingeniería inversa para ActionScript, C++, C#, Delphi, Java.
- Lenguajes de programación Python, PHP[10], VB.NET y Visual Basic.
- Soporte para MDA (transformaciones).
- Validación de modelos.
- Soporte para métricas.
- Soporte para elementos fuera de diagramas (por ejemplo requisitos).
- Generación de documentación. [10]

Tipos de Diagramas que realiza:

Diagramas en Enterprise Architect.	
Diagramas de Tiempo.	Diagrama de Clases.
Diagrama de Casos de Uso.	Diagrama de Objetos.
Diagrama de Actividades.	Diagrama de Despliegue.
Diagrama de Comunicación.	Diagrama de Paquetes.
Diagrama de Interacción.	Diagrama de Composición.
Diagrama de Estado.	Diagrama de Componentes.
Diagrama de Secuencia.	[10]

Tabla 3: Diagramas Enterprise Architect.

ArgoUML

Es una aplicación de diagramas de UML escrita en Java y publicada bajo la Licencia BSD[11]. Fácil de usar, interactivo, posee un entorno gráfico de diseño de software, que soporta el diseño, desarrollo y documentación de las aplicaciones de software orientado a objetos. La ventana de ArgoUML se divide en cuatro secciones: un explorador de elementos, una lista, un panel de detalles con pestañas y el área de dibujo. [11]

Funcionalidades:

- Soporta los estándares abiertos: XMI[12], SVG[13] y PGML[14].

[10] **PHP**: Lenguaje de programación del lado del servidor

[11] **Licencia BSD:** otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*). Es una licencia de software libre permisiva y permite el uso del código fuente en software no libre.

[12] **XMI**: Estándar para la exportación y exportación de archivos UML.

- Es 100% independiente de la plataforma gracias a la utilización exclusiva de Java.
- Permite la ampliación o personalización con Open Source.
- Posee funciones cognitivas tales como: la reflexión en la acción, el diseño de oportunistas, la comprensión y resolución de problemas.
- Multiplataforma.
- Desarrollado en Java.
- Provee todas las funcionalidades para modelar bajo UML.
- Genera código en varios lenguajes.
- Sus diseños son exportables a XMI y pueden ser importados por algunos Frameworks.
- Tiene un depurador del diseño que vamos creando, el depurador sugiere soluciones o detecta incongruencias, sus mensajes son bastante claros y de mucha ayuda.
- Exportación a seis formatos gráficos.
- Documentación y comunidad de usuarios. [11]

Tipos de Diagramas que realiza:

Diagramas en ArgoUML.	
Diagrama de Casos de Uso.	Diagrama de Clases.
Diagrama de Secuencia.	Diagrama de Colaboración.
Diagrama de Estado.	Diagrama de Actividades.
Diagrama de Despliegue.	Diagrama de Interacción. [11]

Tabla 4: Diagramas ArgoUML.

STARUML

StarUML es una herramienta para el modelamiento de software basado en los estándares Lenguaje de Modelado Unificado (UML) y Model Driven Arquitecture (MDA). [12]

Funcionalidades:

- Soporte completo al diseño UML.
- Define elementos propios para los diagramas, que no necesariamente pertenezcan al estándar de UML.
- Posee la capacidad de generar código a partir de los diagramas y viceversa, funcionando para los lenguajes c++, c# y java.
- Genera documentación en formatos Word, Excel y PowerPoint sobre los diagramas.
- Posee patrones GoF (Gang of Four), EJB (Enterprise JavaBeans) y personalizados.
- Contiene plantillas de proyectos.

[13] **SVG:** (Gráficos Vectoriales Redimensiónales) Especificación para describir gráficos bidimensionales.

[14] **PGML**: Lenguaje de Apunte de Gráficos de Precisión. Para gráficos vectoriales.

- Provee la posibilidad de crear plugins para el programa. [12]

Tipos de Diagramas que realiza:

Diagramas en STARUML.	
Diagrama de casos de uso.	Diagrama de clase.
Diagrama de secuencia.	Diagrama de colaboración.
Diagrama de estados.	Diagrama de actividad.
Diagrama de componentes.	Diagrama de despliegue.
Diagrama de composición estructural.	Diagrama de interacción. [12]

Tabla 5: Diagramas en STARUML.

Umbrello UML

"Umbrello UML Modeller es una herramienta de diagramas" que ayuda en el proceso del desarrollo de software y facilita la creación de un producto de alta calidad. Puede agrupar varios diagramas relacionados en un solo fichero XMI, estos estarán organizados en diferentes vistas (lógica, de casos de uso, de componentes, etc.). La creación de un tipo de diagrama está restringida a un tipo de vista determinado. [13]

Funcionalidades:

- Umbrelo UML Modeller incluye soporte para los siguientes lenguajes: C++, Java, C#, D, PHP, JavaScript, ActionScript, SQL, Pascal, Ada, Python, IDL, XML Schema, Perl, Ruby, Tcl.
- Posee tres áreas, estas áreas reciben el nombre de: Vista en árbol, Área de trabajo, Ventana de documentación.
- Posee generación de código: Puede generar código fuente en varios lenguajes de programación, a partir de la maqueta UML para ayudar a comenzar la implementación de su proyecto. El código generado consta de declaraciones de clases con sus métodos y atributos, de forma que se pueda "rellenar los espacios en blanco" proporcionando la funcionalidad de las operaciones de sus clases.
- Posee importación de código: Puede importar código fuente de sus proyectos actuales para ayudarle a crear los esquemas de sus sistemas. [13]
- Copia de objetos como imágenes PNG: Además de ofrecer la funcionalidad normal de copiar, cortar y pegar que se espera en la administración de los objetos de los diferentes diagramas, Umbrello UML Modeller puede copiar los objetos como imágenes PNG de forma que pueda insertarlos en cualquier otro tipo de documento.
- Exportar como imagen: Puede exportar como imagen un diagrama completo.
- Impresión: Permite imprimir diagramas individuales.
- Carpetas lógicas: Para organizar mejor la maqueta, especialmente en los proyectos grandes, es posible crear carpetas lógicas en la vista de árbol. [14]

Tipos de Diagramas que realiza:

Diagramas en Umbrello UML.	
Diagrama de casos de uso.	Diagrama de componentes.
Diagrama de despliegue.	Diagrama de modelo entidad-relación.
Diagrama de clases.	Diagrama de secuencia.
Diagrama de estados.	Diagrama de actividades.
Diagrama de colaboración.	Diagrama de interacción. [14]

Tabla 6: Diagramas Umbrello UML.

BoUML

La parte gráfica de BOUML se basa en Qt y el modelo se organiza en vistas y opcionalmente en paquetes. Una vez creada una vista, se pueden añadir sus elementos, generalmente a través de diagramas. BOUML no permite añadir elementos de forma arbitraria.

Funcionalidades:

- Puede ser redistribuido y/o modificado bajo los términos de la Licencia Pública General (General Public License, GNU).
- Permite especificar y generar código Java, C++, IDL, PHP y Python.
- Es multiplataforma: Linux, Windows, MacOS.
- Es rápida y no consume mucha memoria.
- Soporta gran cantidad de diagramas.
- Es extensible: los generadores de código son extensiones (“plug-outs”) incluidas en la distribución.
- Es capaz de generar documentación en varios formatos (HTML, XMl).

Tipos de diagramas que realiza

Diagramas en BOUML.	
Diagrama de casos de uso.	Diagrama de componentes.
Diagrama de despliegue.	Diagrama de objeto.
Diagrama de clases.	Diagrama de secuencia.
Diagrama de estados.	Diagrama de actividades.
Diagrama de colaboración.	Diagrama de interacción. [41]

Tabla 7: Diagramas en BOUML.

Gaphor

Gaphor es una herramienta fácil de usar en entorno de modelado. Es capaz de crear buenos diagramas UML para la documentación y para las decisiones de diseño. Gaphor ayuda a crear aplicaciones. [15]

Funcionalidades:

- Es compatible con otras herramientas de modelado.
- Puede definir los tipos de estereotipos de clase.

- Posee clases de apoyo a los atributos estereotipados.
- Exporta los diagramas en SVG[15], PNG, PDF.
- Es multiplataforma: funciona tanto en Unix y Windows.
- Posee un formato de archivo estable: garantiza que los archivos antiguos sean legibles por Gaphor.
- Posee infraestructura API[16] permite que Gaphor sea fácil de extender. [16]

Tipos de Diagramas que realiza:

Diagramas en Gaphor.	
Diagramas de clases.	Diagramas de componentes.
Diagramas de casos de uso.	Diagramas de estado.
Diagramas de interacción.	Diagramas de perfiles.
Diagrama de secuencia.	Diagrama de colaboración.
Diagramas de esquemas de acción. [16]	

Tabla 8: Diagramas Gaphor.

[15] **SVG**: Gráficos vectoriales redimensionales.

[16] **Interfaz de programación de aplicaciones** (**API**): es el conjunto de funciones y procedimientos se utilizan por otro software como una capa de abstracción. Son usadas generalmente en las librerías.

2.2 Tabla comparativa de funcionalidades

Para la siguiente tabla se tiene en cuenta las herramientas en cuanto a las funcionalidades.

Herramientas	**Funcionalidades**																								
	Notaciones																				**IDEF**	**Modelos de BD**			
	UML												**BPMN**												
	CU	**A**	**C**	**S**	**CL**	**D**	**I**	**CN**	**O**	**E**	**P**	**CS**	**PN**	**Co**	**FD**	**EPC**	**MP**	**Or**	**WSDL**	**H**		**Relacional**	**Jerárquico**	**Orientado a objeto**	**Red**
Visual Paradimg	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-	X	-
Rational Rose	X	X	X	X	X	X	X	X	-	-	X	-	-	-	-	-	-	-	-	-	X	-	-	-	-
Enterprise Architect	X	X	X	X	X	X	X	X	X	X	X	X	X	-	-	-	-	-	-	-	X	X	-	-	-
ArgoUML	X	X	X	X	X	X	X	-	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
STARUML	X	X	X	X	X	X	X	X	-	X	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-
Umbrello UML	X	X	X	X	X	X	X	X	-	X	-	-	-	-	-	-	-	-	-	-	-	X	-	-	-
Gaphor	X	-	X	X	X	-	X	X	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
BoUML	X	X	X	X	X	X	X	X	X	X															
Suite de IGSW	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabla 9: Herramientas de modelado de software.

Esta tabla se realiza para analizar las funcionalidades de cada una de las herramientas CASE de modelado de software estudiadas, para así determinar los requisitos y funcionalidades con los que cuenta la herramienta a diseñar.

Leyenda: La (X) representa la presencia de la funcionalidad en la herramienta, y el (-) que no aparece la funcionalidad en la herramienta.

CU: Diagrama de caso de uso.	A: Diagrama de actividades.
C: Diagrama de clases.	S: Diagrama de secuencia.
CL: Diagrama de colaboración.	D: Diagrama de despliegue.
I: Diagrama de interacción.	CS: Diagrama de composición.
O: Diagrama de objetos.	E: Diagrama de estados.
P: Diagrama de paquetes.	CN: Diagrama de componentes.
IDEF: Métodos definición integrado.	Co: Diagrama de conversación.
PN: Diagrama de proceso de negocio.	Or: Diagrama de organigramas.
FD: Diagrama de flujo de datos.	WSDL: Diagrama de WSDL.
MP: Diagrama de mapa de procesos.	H: Diagrama de hechos.
EPC: Diagramas de cadenas de procesos condicionados por eventos.	

Tabla 10: Diagramas.

Luego del estudio y análisis de las herramientas anteriores se concluye que el Visual Paradimg es la herramienta candidata para el modelado de software ya que cuenta con todas las funcionalidades especificadas anteriormente en la tabla 9. Esta es una herramienta privativa que no cumple con los requisitos legales pues se pagan las licencias para poder hacer uso de ella para fines comerciales. Por lo que se escoge la herramienta Gaphor para personalizar, puesto que esta es una herramienta libre que está implementada en Python, teniendo en cuenta que se puede reutilizar su código fuente. Debido a que este es el lenguaje a utilizar en el desarrollo de la suite de ingeniería de software, por los beneficios que provee.

3. Arquitectura de Herramientas CASE de modelado de software.

En este epígrafe se analizan los diferentes estilos arquitectónicos, los patrones de diseño y los patrones arquitectónicos existentes.

3.1 Estilos Arquitectónicos.

Un estilo arquitectónico es una posición que se adopta a la hora de desarrollar un software definiendo que se debe hacer. Existen diferentes tipos, atendiendo a varias clasificaciones. Los estudiados para este desarrollo son: [17]

Estilos de Flujo de Datos

Esta familia de estilos enfatiza la reutilización y la modificación. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. [17]

- Tubería y filtros: Una tubería (pipeline) es una arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. [17]

Estilos Centrados en Datos

Esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. [17]

- Arquitecturas de Pizarra o Repositorio: En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Un sistema de pizarra se implementa para resolver problemas en los cuales las entidades individuales se manifiestan incapaces de aproximarse a una solución, o para los que no existe una solución analítica, o para los que sí existe, pero es inviable por la dimensión del espacio de búsqueda. [17]

Estilos de Llamada y Retorno

Esta familia de estilos enfatiza la modificación y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. [17]

- Modelo-Vista-Controlador(MVC): El patrón conocido como Modelo-Vista-Controlador(MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- Modelo. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista. Maneja la visualización de la información.
- Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual. [17]

- Arquitectura en Capas: es una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes, y excepto para funciones puntuales de exportación. En algunos sistemas, las capas pueden ser sólo parcialmente opacas. [17]

- Arquitecturas Orientadas a Objetos: Los componentes del estilo se basan en principios Orientado a Objetos (OO): encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. Se puede modificar la implementación de un objeto sin afectar a sus clientes. Asimismo es posible descomponer problemas en colecciones de agentes en interacción. Además un objeto es ante todo una entidad reutilizable en el entorno de desarrollo. [17]
- Arquitecturas Basadas en Componentes: Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. [17]

Estilos de Código Móvil:

Esta familia de estilos enfatiza la portabilidad.

- Arquitectura de Máquinas Virtuales: El estilo comprende básicamente dos formas o sub-estilos, que se han llamado intérpretes y sistemas basados en reglas. Ambas variedades abarcan, sin duda, un extenso espectro que va desde los lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación. [17]

Estilos Peer-to-Peer:

Esta familia, también llamada de componentes independientes, enfatiza la modificación por medio de la separación de las diversas partes que intervienen en la computación. Consiste en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. [17]

- Arquitecturas Basadas en Eventos: Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados en actores, demonios y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. [17]
- Arquitecturas Orientadas a Servicios: Las arquitecturas basadas en servicios se definen como el estilo de procesos distribuidos, que se comunican a través de mensajes. Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas. Los componentes del están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen y la funcionalidad del servicio se puede ampliar o modificar. [17]
- Arquitecturas Basadas en Recursos: Define recursos identificables y métodos para acceder y manipular el estado de esos recursos. [17]

3.2 Patrones de Diseño

Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo de software. [18]

GOF (Pandilla de los 4):

Patrones de creación: Estos patrones crearán objetos de manera que ya no se tienen que instanciar directamente, proporcionando a los programas una mayor flexibilidad para decidir que objetos usar. [19]

- Fábrica (Factory): Este tipo de patrón es muy útil, su objetivo es devolver una instancia de múltiples tipos de objetos, normalmente todos estos objetos provienen de una misma clase padre mientras que se diferencian entre ellos por algún aspecto de comportamiento. El cliente se abstrae de la instancia a crear. Centraliza la creación de objetos y provee una interfaz para el cliente, permitiendo crear una familia de objetos sin especificar su clase. [19]
- Fábrica Abstracta (Abstract Factory): Este patrón añade un nivel más de complejidad, lo que va a hacer es devolvernos diferentes clases Factory según algún parámetro que se le proporcione. Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. Permite proporcionar una biblioteca de clases y sólo se permiten revelar sus interfaces y no sus implementaciones. [19]
- Solitario (Singleton): Es una clase de la que tan sólo puede haber una única instancia. Garantiza que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. [19]
- Constructor (Builder): Separa la construcción y la representación de un objeto complejo, para así permitir que el mismo proceso de construcción sirva para crear diferentes representaciones. Se usa cuando el algoritmo de creación del objeto complejo debe ser independiente, de qué partes y cómo lo componen. Cuando el proceso de construcción debe soportar diferentes representaciones para el objeto que se construye. [19]
- Prototipo (Prototype): Especifica los tipos de objetos a crear usando una instancia prototípica y creando nuevos objetos copiando dicho prototipo. Cuando las clases a instanciar se especifican en tiempo real, o se quiere evitar construir una jerarquía de fábricas paralela a la de productos o las instancias de una clase pueden tomar una de entre unas pocas combinaciones distintas de estado. [19]

Patrones estructurales:

Los patrones estructurales describen como formar estructuras complejas a partir de elementos más simples. Existen dos tipos de patrones de este tipo, de clase y de objeto. [19]

- Adaptador (Adapter): Su finalidad es transformar la interfaz de programación de una clase en otra. Se utilizan adaptadores cuando las clases que no tienen nada que ver funcionen de la misma manera para un programa determinado. El concepto de un Adapter es escribir una nueva clase con la interfaz de programación deseada y hacer que se comunique con la clase cuya interfaz de programación era diferente. [19]

- Puente (Bridge): Se utiliza para separar la interfaz de una clase de su implementación de forma que ambas puedan ser modificadas de manera separada, el objetivo es poder modificar la implementación de la clase sin tener que modificar el código del cliente de la misma. [19]
- Apoderado (Proxy): Lo que hace es cambiar un objeto complejo por otro más simple. Proxy permite posponer su creación hasta que sea realmente necesitado. Un Proxy tiene los mismos métodos que el objeto al que representa pero estos métodos solamente son llamados cuando el objeto ha sido cargado por completo. [19]
- Composición (Composite): Permite a los clientes tratar uniformemente a los objetos simples y compuestos de una estructura jerárquica recursiva. Simplifica notablemente al cliente, al no tener éste que distinguir entre objetos simples y compuestos. Favorece la extensibilidad, ya que es muy fácil añadir nuevos tipos de componentes, tanto simples como compuestos. [19]
- Decorador (Decorator): Añade responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades. Aporta una mayor flexibilidad que la herencia estática, permitiendo, entre otras cosas, añadir una funcionalidad dos o más veces. [19]
- Fachada (FaÇade): Proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. Al separar al cliente de los componentes del subsistema, se reduce el número de objetos con los que el cliente trata, facilitando así el uso del subsistema. [19]
- Peso Mosaca (Flyweight): Mejorar la eficiencia a la hora de mantener una gran cantidad de objetos “de grano fino”, usando la idea de la compartición. Mejora en la eficiencia, sobre todo en relación con el ahorro en el almacenamiento, que aumenta paralelamente con la compartición de objetos. [19]

Patrones de comportamiento: Los patrones de comportamiento fundamentalmente especifican el comportamiento entre los objetos del sistema. [19]

- Acción (Command): Específica una forma simple de separar la ejecución de un comando del entorno que generó dicho comando. Comportamiento, a nivel de objetos. [19]
- Iterador (Iterator): Su función es recorrer un conjunto de datos mediante una interfaz estándar sin tener que conocer los detalles de la implementación de los datos. Además siempre se pueden definir iteradores especiales que realicen algún tipo de acción con los datos o que sólo devuelvan elementos determinados. [19]
- Observador (Observer): Asume que el objeto que contiene los datos es independiente de los objetos que muestran los datos, de manera que son estos objetos los que “observan” los cambios en dichos datos. Al implementar el patrón Observer el objeto que posee los datos se conoce como sujeto, mientras que cada una de las vistas se conocen como observadores. [19]

- Estrategia (Strategy): El patrón Strategy consiste en un conjunto de algoritmos encapsulados en un contexto determinado Context. El cliente puede elegir el algoritmo que prefiera entre los disponibles o puede ser el mismo objeto Context el que elija el más apropiado para cada situación. [19]
- Visitante (Visitor): Este patrón aparenta romper con la programación orientada a objetos en el sentido de que crea una clase externa que va a actuar sobre los datos de otras clases. Representa una operación que está pensada para ser aplicada sobre los elementos de una estructura de objetos, permitiendo así definir y añadir un nuevo comportamiento sin necesidad de cambiar las clases de los elementos de la estructura de objetos. [19]
- Cadena (Chain): Proporciona a más de un objeto la capacidad de atender una petición, para así evitar el acoplamiento con el objeto que hace la petición. Se forma con estos objetos una cadena, en la cual cada objeto o satisface la petición o la pasa al siguiente. Se reduce el acoplamiento, puesto que se libera al objeto que hace la petición de conocer quien se la atiende. Los miembros de la cadena no conocen la estructura entera, sino sólo su sucesor en la misma. [19]
- Mediador (Mediator): El diseño OO enfatiza la distribución de responsabilidades entre objetos. Tal distribución puede devenir en una estructura con muchas conexiones entre objetos. Aunque la distribución favorece la reutilización, la proliferación de relaciones entre objetos puede ser un grave obstáculo. Por ello, este patrón define un objeto que encapsula la forma en que interactúan un grupo de objetos, promoviendo así un acoplamiento débil al evitar las referencias explícitas entre los objetos y permitiendo, por tanto, que su interacción se modifique de forma independiente. [19]
- Plantilla (Template): Define el esqueleto de un algoritmo para una operación, dejando para sus subclases la capacidad de redefinir el funcionamiento de los pasos de este algoritmo, siempre y cuando la estructura del mismo permanezca intacta. Cuando se pretende implementar una sola vez las partes invariantes de un algoritmo, permitiendo a las subclases implementar el comportamiento que puede variar. [19]
- Interprete (Interpreter): Dado un lenguaje, define una representación para su gramática junto con un intérprete que usa dicha representación para interpretar sentencias en ese lenguaje. Facilidad para cambiar o extender la gramática, mediante herencia, dado que las diferentes reglas de la gramática se representan con clases. [19]
- Estado (State): Permitir a un objeto modificar su comportamiento a medida que su estado interno va cambiando, dando así la impresión de que el objeto "cambia de clase". Dado que el comportamiento específico de cada estado está contenido en cada estado concreto, existe una gran flexibilidad para añadir nuevos estados y transiciones, mediante la definición de nuevas subclases. [19]

GRASP: Patrones para asignar responsabilidades:

Patrón Experto: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Durante el diseño orientado a objetos, cuando se definen las interacciones entre los objetos, se toman decisiones sobre la asignación de responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y se presenta la oportunidad de reutilizar los componentes en futuras aplicaciones. [20]

Patrón Creador: Asignarle a una clase B la responsabilidad de crear una instancia de una clase A. El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se conecta con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. [20]

Patrón Bajo acoplamiento: Asignar una responsabilidad para mantener bajo acoplamiento. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. El Bajo Acoplamiento estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. [20]

Patrón Alta cohesión: Asignar una responsabilidad de modo que la cohesión siga siendo alta. En la perspectiva del desafío orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. [20]

Patrón Controlador: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase. Este patrón ofrece una guía para tomar decisiones apropiadas. Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. [20]

Polimorfismo: Como el patrón Experto, el uso del patrón Polimorfismo está acorde al espíritu del patrón. Polimorfismo será el más importante patrón estratégico en el desafío orientado a objetos. Es un principio fundamental en que se fundan las estrategias globales, o planes de ataque, al desafiar como organizar un sistema para que se encargue del trabajo. Un desafío basado en la asignación de responsabilidades mediante el polimorfismo puede ser extendido fácilmente para que realice nuevas variantes. [20]

Fabricación Pura: Para desafiar una Fabricación Pura debe buscarse ante todo un gran potencial de reutilización, asegurándose para ello que sus responsabilidades sean pequeñas y cohesivas. Una fabricación pura suele dividirse atendiendo a su funcionalidad y, por lo mismo, es una especie de objeto de función central. Generalmente se considera que la fabricación es parte de la capa de servicios orientada a objetos de alto nivel en una arquitectura. [20]

Indirección: Se asigna la responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios, y estos no terminen directamente acoplados. El intermediario crea una indirección entre el resto de los componentes o servicios. [20]

No Hables con Extraños: No Hables con Extraños se refiere a no obtener una visibilidad temporal frente a objetos indirectos, que son de conocimiento de otros objetos pero no del cliente. La desventaja de conseguir visibilidad ante extraños es que la solución se acopla entonces a la estructura interna de otros objetos. Ello origina un alto acoplamiento, que hace el diseño menos robusto y más propenso a requerir un cambio si se alteran las relaciones estructurales indirectas. Se asigna la responsabilidad a un objeto directo del cliente para que colabore con un objeto indirecto, de modo que el cliente no necesite saber nada del objeto indirecto. [20]

3.3 Patrones Arquitectónicos

Modelo-Vista-Controlador (MVC): El patrón arquitectónico MVC propone una separación clara entre el modelo y la vista, gracias a un controlador que los mantiene desacoplados. Este modelo es reusable con distintas vistas y nos facilita la utilización de diferentes clientes. Divide una aplicación interactiva en tres componentes. El modelo contiene la información central y los datos. Las vistas despliegan información al usuario. Los controladores capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario. [21]

Patrón en Capas: La capa superior utiliza los servicios definidos por la inferior, pero la capa inferior no es consciente de la capa superior. Además cada capa normalmente esconde los niveles más bajos de las capas por encima. Sólo es necesario conocer la capa superior y es posible despreocuparse de las otras capas por debajo. Se pueden reemplazar capas por implementaciones alternativas de los mismos servicios básicos. Se minimizan las dependencias entre capas y una capa puede ser reutilizada por múltiples servicios de niveles superiores. [22]

Cliente/Servidor: Patrón arquitectónico para el desarrollo de sistemas distribuidos.

• Distribuye una aplicación entre 2 o más componentes especializados cuya ejecución se distribuye entre 1 o más equipos.

• Define dos tipos de entidades diferenciadas (asimétricas) que se responsabilizan de acciones diferentes: clientes y servidores. Especifica 2 tipos de procesos con roles diferenciados.

• Define un modelo de interacción basado en el concepto de servicio implementado sobre un diálogo petición-respuesta. El Cliente inicia el diálogo mediante el envío de peticiones. El Servidor presta el servicio y responde las peticiones recibidas.

• Especifica el modo en que se sincronizan los procesos:

_ Cliente (parte activa) demanda servicios a los servidores se asume que cada petición deberá obtener respuesta diseñado para soportar la interacción con el usuario final.

_ Servidor (parte pasiva) espera las peticiones de los clientes procesa esas peticiones y envía una respuesta diseño orientado a maximizar la eficiencia

• Posibilidad de aplicar el patrón cliente-servidor en múltiples niveles de abstracción dentro de un mismo sistema distribuidos. [23]

Plugin: Una plug-in (extensión) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la principal e interactúa por medio de la Interfaz. Dicho estilo no contiene ninguna variante arquitectónica, se ve de forma independiente. Se enfoca en explicar cómo se puede diseñar una aplicación con el fin de soportar varios plugin permitiendo que la misma se extienda en tiempo de ejecución mediante la carga dinámica de módulos o clases que no conoce durante la compilación.

Desde el punto de vista del desarrollo, un sistema de plugins tiene diversas ventajas:

- Alta independencia entre módulos (también baja cohesión entre módulos). Puesto que cada módulo es, en principio, relativamente independiente de todos los demás es más fácil desarrollar pruebas mediante stubs y drivers sin que se vean afectadas por el resto de componentes de la aplicación.
- Facilidad para extender la aplicación sin necesidad de redistribuir un nuevo ejecutable.
- Si los plugins que desarrollamos son los suficientemente genéricos pueden reutilizarse en otras aplicaciones con lo que el periodo de desarrollo es significativamente menor (el necesario para adaptar el plugin al nuevo sistema).
- Facilidad para adaptarse a cambios en los requisitos añadiendo o modificando funcionalidades mediante la creación o modificación de un plugin adecuado. [24]

Luego del estudio de los estilos y patrones existente se concluye que la propuesta de la arquitectura es 3 capas, la misma debido al encapsulamiento provee aplicaciones más robustas, brinda mantenimiento y soporte más sencillo, pues es mejor hacer cambios en una capa que modificar el código completo, además esta arquitectura permite una separación bien delimitada del código.

Como patrones de diseño se proponen: el creador, pues, este es el encargado de guiar la asignación de responsabilidades específicas a cada una de las clases relacionadas con la creación de objetos, brindando menos dependencias entre las clases y permitiendo la reutilización de código. El experto, asigna la responsabilidad a la clase que cuenta con la información necesaria para cumplir con la misma, además conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide y esto permite tener sistemas de fácil mantenimiento. El bajo acoplamiento permite que no haya tanta dependencia entre las clases, esto reduce el impacto de los cambios, y además permite que sean más reutilizables las clases. El alta cohesión, caracteriza a las clases con responsabilidades estrechamente relacionadas para que no realicen un trabajo enorme.

4. Notaciones UML y BPMN

Las notaciones UML y BPMN son las que se usan para la realización de los diagramas de modelado por lo que se hace un breve estudio de las mismas.

4.1 Definición de UML.

El UML es un lenguaje de modelado cuyo vocabulario y sintaxis están ideados para la representación conceptual y física de un sistema. Sus modelos son precisos, no ambiguos, completos y pueden ser trasladados directamente a una gran variedad de lenguajes de programación, como Java, C++ o Visual Basic, pero también a tablas de bases de datos relacionales y orientados a objetos. Es posible generar código a partir de un modelo UML (ingeniería directa) y también puede construirse un modelo a partir de la implementación (ingeniería inversa). [25]

El UML soporta un conjunto rico en elementos de notación gráficos. También soporta la idea de extensiones personalizadas a través de elementos estereotipados. Provee beneficios significativos para los ingenieros de software y las organizaciones al ayudarles a construir modelos rigurosos, trazables y mantenibles, que soporten el ciclo de vida de desarrollo de software completo. [26]

El Lenguaje Unificado de Modelado prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. [27]

4.2 Diagramas que admite la notación UML. Caracterización.

Diagrama de Casos de Uso: Casos de Uso es una técnica para capturar información de cómo un sistema o negocio trabaja, o de cómo se desea que trabaje. No pertenece estrictamente al enfoque orientado a objeto, es una técnica para captura de requisitos. [28]

- Actores: Un actor es un usuario del sistema. Un actor usa un caso de uso para ejecutar una porción de trabajo de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define rol en el sistema y el alcance de su acción.
- La especificación formal de un caso de uso incluye 3 elementos básicos:
  - ✓ Requisitos. Son los requisitos funcionales formales que el caso de uso debe proveer al usuario final. Ellos corresponden a las especificaciones funcionales de las metodologías estructuradas. Un requisito es un contrato del cual el caso de uso realizará alguna acción o proveerá algún valor al sistema.
  - ✓ Restricciones. Estas son las reglas formales y las limitaciones bajo las que opera un caso de uso e incluyen las pre-condiciones, las post-condiciones y las invariantes.
  - ✓ Escenarios. Son descripciones formales del flujo de eventos que ocurre durante una instancia de un caso de uso. Usualmente se describen con texto y corresponden a una representación textual del diagrama de secuencia.
- Incluye y Extiende:

Un caso de uso puede incluir la funcionalidad de otro como parte de su procesamiento normal. Generalmente se asume que los casos de uso incluidos se llamarán cada vez que se ejecute el camino base. Un caso de uso puede ser incluido por uno o más casos de uso, ayudando así a reducir la duplicación de funcionalidad al dividir el comportamiento común en los casos de uso que se reutilizan muchas veces. Un caso de uso puede extender el comportamiento de otro caso de uso; típicamente cuando ocurren situaciones excepcionales. [26]

Diagramas de Actividad: El diagrama de actividad es un diagrama de flujo del proceso multi-propósito que se usa para modelar el comportamiento del sistema. Los diagramas de actividad se pueden usar para modelar un caso de uso, una clase, o un método complicado. [27]

Diagrama de Clases: El diagrama de clases es el diagrama principal para el análisis y el diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones. [28]

Diagramas 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. Un diagrama de secuencia representa típicamente un único escenario de caso de uso o flujo de eventos. Los diagramas de secuencia muestran el flujo de mensajes de un objeto a otro y, como tales, representan los métodos y los eventos soportados por un objeto/clase. [26]

Diagramas de Colaboración: El diagrama de colaboración se centra en estudiar todos los efectos de un objeto dado durante un escenario. Los objetos se conectan por medio de enlaces, cada enlace representa una instancia de una asociación entre las clases implicadas. El enlace muestra los mensajes enviados entre los objetos, el tipo de mensaje (sincrónico, asincrónico, simple, blanking, y 'time-out'), y la visibilidad de un objeto con respecto a los otros. [27]

Diagrama de Implementación: Un diagrama de implementación se asocia típicamente con un caso de uso para documentar qué elementos de diseño (por ejemplo, componentes y clases) implementará la funcionalidad del caso de uso en el nuevo sistema. Esto provee un alto grado de trazabilidad al diseñador, al cliente y al equipo que construirá el sistema. [26]

Diagrama de Componentes: Muestra la organización y dependencias de un conjunto de componentes. Cubren la vista de implementación estática de un sistema. Un componente es un módulo de código, de modo que los diagramas de componentes son los análogos físicos a los diagramas de clases. Muestran como está organizado un conjunto de componentes y las dependencias que existen entre ellos. [25]

Diagrama de Despliegue: Muestra la configuración del hardware del sistema, los nodos de proceso y los componentes empleados por éstos. Cubren la vista de despliegue estática de una arquitectura. Los diagramas de despliegue sirven para modelar la configuración del hardware del sistema, mostrando qué nodos lo componen. [25]

4.3 Definición de BPMN

BPMN (Notación para el Modelado de Procesos de Negocio): Es una notación a través de la cual se expresan los procesos de negocio en un diagrama de procesos de negocio (BPD). Este estándar agrupa la planificación y gestión del flujo de trabajo, así como el modelado y la arquitectura.

Proporciona un lenguaje gráfico común, con el fin de facilitar su comprensión a los usuarios de negocios. Integra las funciones empresariales. Utiliza una Arquitectura Orientada por Servicios (SOA), con el objetivo de adaptarse rápidamente a los cambios y oportunidades del negocio. Combina las capacidades del software y la experiencia de negocio para optimizar los procesos y facilitar la innovación del negocio. [25]

Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. [26]

4.4 Diagramas que admite la notación BPMN. Caracterización.

Los elementos en BPMN se encuentran clasificados dentro de 4 categorías:

Objetos de Flujo: son los principales elementos gráficos que definen el comportamiento de los procesos. Dentro de los objetos de Flujo encontramos: [29]

- ✓ Eventos: Suceden durante el curso de un proceso de negocio, afectan el flujo del proceso y usualmente tienen una causa y un resultado. Se usa inicio, fin y temporizador, estos elementos son eventos y a su vez se encuentran clasificados en 3 tipos: eventos de inicio, eventos intermedios, eventos de fin. [29]
- ✓ Actividades: Estas representan el trabajo que es ejecutado dentro de un proceso de negocio. Las actividades pueden ser compuestas o no, por lo que los dos tipos de actividades existentes: tareas y subprocesos. [29]
- ✓ Compuertas: Son elementos del modelado que se utilizan para controlar la divergencia y la convergencia del flujo. Existen 5 tipos de compuertas: compuerta exclusiva, compuerta basada en eventos, compuerta paralela, compuerta inclusiva, compuerta compleja. [29]

Objetos de Conexión: Son los elementos usados para conectar dos objetos del flujo dentro de un proceso. Existen 3 tipos de objetos de conexión: líneas de secuencia, asociaciones, líneas de mensaje. [29]

Canales: son elementos utilizados para organizar las actividades del flujo en diferentes categorías visuales que representan áreas funcionales, roles o responsabilidades: pools y lanes. [29]

Artefactos: Los artefactos son usados para proveer información adicional sobre el proceso. Existen 3 tipos: objetos de datos, grupos, anotaciones. [29]

Los diagramas que admite esta notación son:

Diagrama de Proceso de Negocio: Se utiliza para modelar procesos privados o públicos de una organización. Muestra cómo y quién efectúa las actividades que generan valor para la organización. [30]

Diagrama de conversación: Se utiliza para modelar conversaciones inter-organizacionales. [30]

Diagrama Mapa de Procesos: Es un diagrama de valor; un inventario gráfico de los procesos de una organización. [31]

Diagrama de Flujo de Datos: Es un modelo lógico-gráfico, que ayuda a representar el funcionamiento de un sistema, este permite incorporar opciones para el depurado de algoritmos, facilitando la localización de errores de ejecución y lógicos más habituales. [32]

Diagrama EPC: Una CPE se crea agrupando los eventos con las funciones en una cadena y muestra el curso cronológico de un proceso de negocios. [33]

Organigrama: El organigrama es la forma típica de representación de estructuras organizativas. Un esquema de esta clase refleja las unidades organizativas, representaciones de conjunto de personas que llevan a cabo tareas para alcanzar los objetivos de negocios, y sus interrelaciones, dependiendo del criterio de estructuración seleccionado. [33]

5. Infraestructura tecnológica para el desarrollo del módulo de modelado para la suite de Ingeniería de Software.

En este epígrafe se explica la infraestructura tecnológica definida por el proyecto a utilizar en el desarrollo de este trabajo.

5.1 Metodología de Desarrollo de software.

Las Metodologías Ágiles (MA) valoran al individuo y las interacciones en el equipo de desarrollo, la colaboración con el cliente y la respuesta a los cambios. La prioridad principal de las MA es satisfacer al cliente mediante tempranas y continuas entregas de software que le reporte un valor y que funcione, con el menor intervalo de tiempo posible entre una entrega y la siguiente. La atención continua a la calidad técnica y al buen diseño mejora la agilidad. En intervalos regulares, el equipo reflexiona respecto de cómo llegar a ser más efectivo, y según esto ajusta su comportamiento. [34]

5.1.1 OpenUp

La metodología OpenUp se basa en proceso unificado iterativo e incremental centrado en el desarrollo de software colaborativo. Está basado en casos de uso, la gestión del riesgo, y una arquitectura enfocada a impulsar el desarrollo. [28] OpenUp es además un marco de trabajo para procesos de desarrollo de software. [29]

Cuenta con 4 fases: Inicio, Elaboración, Construcción y Transición. En este trabajo se llega hasta la fase de elaboración debido a que el alcance del mismo concluye en la definición y evaluación de la arquitectura.

En la fase de Inicio se define el propósito, alcance y los objetivos del proyecto, además de determinar si se debe continuar o cancelar en dependencia de la información obtenida. El objetivo de ésta fase es capturar las necesidades de los stakeholder en los objetivos del ciclo de vida para el proyecto.

Los cuatro objetivos de esta fase que clarifican el alcance, objetivos del proyecto y factibilidad de la solución pretendida son:

- ✓ Entender lo que se va a construir, determinando una visión general e incluyendo el alcance del sistema y sus límites.
- ✓ Identificar las funcionalidades claves del sistema, decidiendo qué requerimientos son más críticos.
- ✓ Determinar por lo menos una posible solución, evaluando si la visión es técnicamente factible. Esto puede involucrar identificar una arquitectura candidata de alto nivel, realizar prototipos técnicos, o ambas cosas.
- ✓ Entender a un alto nivel la estimación de costos, calendario y riesgos asociados al proyecto.

La fase Elaboración es donde se tratan los riesgos estructuralmente significativos para la arquitectura. El propósito de esta fase es establecer la línea base de la arquitectura del sistema y proveer una base estable para la mayor parte del esfuerzo de desarrollo de la siguiente fase.

Los riesgos asociados con los requerimientos, arquitectura, costos y calendario son tratados mediante los siguientes objetivos:

- ✓ Obtener un entendimiento más detallado de los requerimientos, asegurándose de tener un conocimiento profundo de los requerimientos más críticos.
- ✓ Diseñar, implementar, validar y establecer la línea base de la arquitectura (para el esqueleto de la estructura del sistema).
- ✓ Mitigar riesgos esenciales y producir un calendario apropiado y estimación de costos.

Esta metodología tiene diferentes disciplinas, dentro de las cuales se presentan varias actividades que generan diversos artefactos. Por las condiciones de este trabajo solo se utilizaran las siguientes disciplinas: Requerimientos y Arquitectura, además de las disciplinas de apoyo Administración de la configuración y cambio, Administración de proyecto y los artefactos que se generan son los del programa de mejora[17].

Disciplinas	**Tareas**	**Artefactos**
Requerimientos.	• Identificar y resaltar requerimientos. • Detallar Escenarios de Caso de Uso. • Detallar requerimientos generales del sistema. • Desarrollar una visión técnica.	• Reporte de trazabilidad. • Glosario de términos • Especificación de Requisitos. • Descripción de los requisitos.

[17] **PM**: Marco de trabajo homogéneo para todos los proyectos, para la reducción de costos y errores y el aumento de la productividad.

Arquitectura.	• Refinar la arquitectura. • Visualizar (Preveer) la arquitectura.	• Arquitectura Vista de casos de uso. • Arquitectura Vista de Sistema. • Arquitectura Vista de Implementación. • Arquitectura Vista de Despliegue.
Administración de Proyecto.	• Evaluar resultados. • Administrar La iteración. • Planear la iteración. • Planear el proyecto. • Solicitar cambios.	• Plan de desarrollo de software. • Proyecto técnico. • Plan de pruebas.

Tabla 11: Relación entre las disciplinas, tareas y artefactos de la metodología OpenUp.

5.2 Lenguajes de programación

5.2.1. Python

Python es un lenguaje de programación interpretado, orientado a objetos, multiplataforma y de sintaxis sencilla. Permite dividir su programa en módulos reutilizables desde otros programas en Python. El lenguaje incorpora una gran colección de módulos estándar que se pueden utilizar como base de los programas. Permite escribir programas muy compactos y legibles, con muy pocas líneas de código se pueden lograr diversas funcionalidades. La razón de esto es que Python usa tabulación o (espaciado) para mostrar estructura de bloques. Permite el desarrollo de tareas científicas, en los que hay que simular y prototipar rápidamente. [35]

5.3 Herramientas CASE

5.3.1. Eclipse

La plataforma Eclipse consiste en un Entorno de Desarrollo Integrado (IDE, Integrated Development Environment) abierto y extensible, cuenta con numerosas herramientas de desarrollo de software. También da soporte a otros lenguajes de programación, como son C/C++, Cobol, Fortran, PHP o Python. [36] Es una plataforma abierta para la integración de herramientas de desarrollo construidas por una comunidad abierta y proveedores de herramientas. Trabaja y se desarrolla con los criterios de código abierto y licencia libre. [37] Dispone de un Editor de texto y la compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, sobre el cual se pueden montar herramientas de desarrollo para cualquier lenguaje mediante la implementación de los plugins adecuados como comando para la realización de diagramas UML generando código. [38]

5.3.2. QT Designer

Qt Designer es una herramienta visual para diseñar y construir interfaces gráficas para usuarios (GUIs) de componentes Qt. Los dispositivos y las formas creadas con Qt Designer se integran como una sola pieza

con código programado, utilizando Qt y el mecanismo de ranuras, esto permite asignarle el comportamiento a los elementos gráficos. Todas las propiedades de Qt Designer puede cambiar dinámicamente dentro del código. [39]

5.3.3. Visual Paradigm

Visual Paradigm es una herramienta para la construcción de sistemas a gran escala. Ofrece alta confiabilidad y estabilidad en el desarrollo. Es una de las herramientas más completas para el modelado de software, que brinda varias funcionalidades útiles al desarrollador. [40]

Visual Paradigm también ofrece:

- Navegación intuitiva entre la escritura del código y su visualización;
- Potente generador de informes en formato PDF/HTML;
- Documentación automática Ad-hoc;
- Ambiente visualmente superior de modelado;
- Sofisticado diagramador automáticamente de layout;
- Sincronización de código fuente en tiempo real. [40]

Conclusiones parciales

En este capítulo se realiza un estudio de las herramientas de modelado de software con el objetivo de sentar las bases, para el diseño del módulo de modelado de software. Se determina la herramienta a personalizar, además de las funcionalidades con las que esta cuenta. También se definen las herramientas, metodología de desarrollo y patrones arquitectónicos y de diseño a usar, lográndose en su totalidad el objetivo de este capítulo.

Capítulo #2: Obtención de Requisitos y Diseño de la arquitectura de la herramienta de modelado para la Suite de Ingeniería de Software

Introducción

Este capítulo se expresa en función de las disciplinas de la metodología de desarrollo. Se recogen los aspectos de la arquitectura basado en la metodología antes mencionada, para lograr la creación de una aplicación robusta. Para ello se define la arquitectura del módulo de modelado teniendo en cuenta los requisitos funcionales y no funcionales de la misma y las vistas arquitectónicas fundamentales desde las que se puede apreciar el módulo de modelado de software.

1. Requisitos.

1.1. Requisitos del sistema

Requisitos funcionales:

A continuación se muestran los requisitos más importantes para el sistema, agrupados por casos de uso:

CU1 Iniciar Proyecto de Modelado en blanco.

RF_2.1 Crear un proyecto de modelado en blanco.

CU2 Iniciar Proyecto de Modelado ya existente.

RF_2.2 Crear un proyecto de modelado a partir de uno que existe en formato XMI.

RF_2.3 Crear un proyecto de modelado a partir de uno creado en Rational Rose.

CU3 Cerrar proyecto de modelado.

RF_3 Cerrar proyecto de modelado.

CU4 Eliminar proyecto de modelado.

RF_4 Eliminar proyecto de modelado.

CU5 Guardar proyecto de modelado.

RF_7.2 Guardar proyecto de modelado en formato plantilla.

CU6 Importar proyecto de modelado.

RF_8.2 Importar proyecto modelado en formato XML.

RF_8.3 Importar proyecto modelado en Rational Rose.

RF_8.4 Importar proyecto modelado en Visual Paradigm.

CU7 Gestionar diagramas

RF_11 Crear diagramas

RF_11.1 Crear diagramas UML.

RF_11.1.1 Crear diagramas CU.

RF_11.1.2 Crear diagramas DA.

RF_11.1.3 Crear diagramas Clases.

RF_11.1.4 Crear diagramas Colaboración.

RF_11.1.5 Crear diagramas Secuencia.

RF_11.1.6 Crear diagramas de Interacción.

RF_11.1.7 Crear diagramas Despliegue.

RF_11.1.8 Crear diagramas de Componentes.

RF_11.1.9 Crear diagramas de Objetos.

RF_11.1.10 Crear diagramas de Estados.

RF_11.1.11 Crear diagramas de Paquetes.

RF_11.1.12 Crear diagramas de Composición.

RF_11.2 Crear diagramas BPMN.

RF_11.2.2 Crear diagramas de conversación.

RF_11.2.3 Crear diagramas de flujo de datos.

RF_11.2.4 Crear diagramas EPC[18].

RF_11.2.5 Crear diagramas de mapa de procesos.

RF_11.2.6 Crear diagramas de organigrama.

RF_11.2.7 Crear diagramas de WSDL[19].

RF_11.2.8 Crear diagramas de hechos.

CU8 Manipular diagrama.

RF_16.1 Copiar estereotipo.

RF_16.2 Cortar estereotipo.

RF_16.3 Pegar un estereotipo.

RF_16.4 Redimensionar estereotipo.

RF_16.5 Crear estereotipo.

RF_16.6 Eliminar estereotipo.

RF_16.7 Nombrar estereotipo.

RF_16.8 Modificar estereotipo.

RF_16.9 Visualizar estereotipo.

RF_16.10 Relacionar estereotipos.

Para más detalles ver Plantilla 0113_Especificación de Requisitos de Software.doc del proyecto Nova-QALIT.

Requisitos no funcionales

Usabilidad

RNF_1. Los usuarios finales de la Suite de IS son desarrolladores de software.

[18] **EPC**: Cadenas de procesos condicionados por eventos.

[19] **WSDL**: Permite describir la interfaz de un servicio web en un formato XML.

La aplicación va a ser utilizada por desarrolladores de software, que pueden tener niveles de conocimientos sobre esta área: medio, bajo o avanzado.

RNF_2. La Suite de IS es una aplicación de escritorio.

Debido a que se requiere de un software que sea rápido, que permita realizar modelos con diferentes niveles de complejidad, el ingreso de grandes cantidades de datos, seguridad, y la continuidad del trabajo aún cuando internet esté ausente.

Eficiencia

RNF_3. Para dar cumplimiento a los requisitos del sistema las transacciones deben tener una duración de 15 a 60 segundos.

Las transacciones que ocurren en el momento de ejecutar las operaciones deben tener una duración no mayor de 15 segundos y cómo máximo 60 segundos garantizando la respuesta rápida del sistema a las necesidades de los usuarios.

RNF_4. En las PC clientes el sistema debe consumir entre el 5% y el 40% de memoria.

En las PC clientes el sistema debe consumir entre el 5 y el 40% de memoria teniendo en cuenta que estas deben tener un tamaño de 512MB.

Soporte

RNF_5. Si se encuentran no conformidades en el módulo de modelado estas deben ser corregidas en un período máximo de un mes.

Restricciones de diseño

RNF_6. La aplicación debe hacerse utilizando el lenguaje de programación Python 3.3.0.

RNF_7. Para cumplir con los requisitos de diseño y creación de gráfico de la suite de IS se va a utilizar la herramienta QT Designer 4.7.2.

RNF_8. Para la codificación de la herramienta se va a utilizar el IDE (Entorno Integrado de Desarrollo) Eclipse en su versión Galileo.

RNF_9. Para modelar la suite de IS se debe utilizar la herramienta Visual Paradigm 10.1.

Componentes Comprados

RNF_10. El módulo de modelado de software tiene como base para su desarrollo la herramienta Gaphor en su versión 0.17.1.

La herramienta Gaphor permite el modelado de software de una manera simple a través de la notación UML, la misma está desarrollada utilizando el lenguaje de programación Python y garantiza la comunicación con el usuario mediante la biblioteca GTK. Está liberada con la GNU General Public License (GPL).

Interfaz

Interfaz de Hardware

RNF_11. El sistema debe funcionar en hardware con bajas prestaciones.

El hardware con requerimientos mínimos sobre el que debe funcionar la Suite de IS se caracteriza por: las pc clientes Micro Celeron 266, RAM 256MB y servidor de datos PostgreSQL en su versión 9.1.

Interfaces Software

RNF_12. La aplicación debe funcionar sobre los sistemas operativos Linux y Windows, recomendándose utilizar Ubuntu 10.4 o superior o Nova 2011 o superior o Windows NT o superior.

Interfaz de comunicación

RNF_13. Se debe tener implementado en las estaciones de trabajo o servidores redes LAN (Cableadas o Inalámbricas).

Estándares Aplicables

RNF_14. La documentación técnica generada del desarrollo de software debe quedar guardada en los artefactos que provee el expediente de proyecto en su versión 3.3 propuesto por el programa de mejora emitido por CaliSoft.

RNF_15. El estándar de codificación que se va a utilizar es PEP 8.

PEP 8 proporciona las convenciones de codificación para el código Python que comprende la biblioteca estándar de la distribución principal de Python, la utilización de este estándar permite que los programadores puedan hablar utilizando un lenguaje común, lo cual apoya la comunicación y continuidad del trabajo.

Para más detalles ver Plantilla 0113_Especificación de Requisitos de Software.doc del Nova-QALIT.

2. Descripción de los requisitos.

De un total de 24 descripciones de requisitos se escoge Comenzar Proyecto de Modelado para mostrar la manera en que este fue detallado para un mejor entendimiento por parte de los desarrolladores de software.

Comenzar Proyecto de Modelado:

Permite crear un proyecto de modelado en blanco o cargar uno ya existente que puede estar hecho en Rational Rose o en formato XMI.

Precondiciones	Para crear un nuevo proyecto de modelado debe estar abierta la herramienta.
Flujo de eventos	
Flujo Básico: "Comenzar Proyecto de Modelado"	
1	Seleccionar la opción "Comenzar Proyecto de Modelado" en la siguiente ruta de la barra de menú, "Módulos/Modelado".
2	Muestra las opciones para comenzar un proyecto de modelado.
3	Seleccionar la opción que desee: • Crear Proyecto de Modelado en Blanco. Ver Sección 1: "Crear Proyecto de Modelado en Blanco". • Crear Proyecto de Modelado desde Rational Rose. Ver Sección 2: "Crear Proyecto de Modelado desde Rational Rose". • Crear Proyecto de Modelado desde formato XMI. Ver Sección 3: "Crear Proyecto de Modelado con formato XMI".
Pos-condiciones	

1	No Procede
Flujos alternativos	
Flujo alternativo <<Nº Evento>>.<<letra iniciando por la a>> <Condición que dio lugar a la extensión>	
1	No procede
Pos-condiciones	
1	No procede
Sección 1: "Crear Proyecto de Modelado en Blanco"	
Flujo Básico	
1.	Selecciona la opción "Proyecto de Modelado en Blanco" en la siguiente ruta de la barra de menú, "Módulos/Modelado/Comenzar Proyecto de Modelado".
2.	Muestra una ventana con las opciones para comenzar un nuevo proyecto de modelado.
3.	Introducir los siguientes datos Nombre del Proyecto, Notación de modelado, Lenguaje de programación, Desarrollador, Compañía, Tipo de software y Descripción del proyecto.
4.	Presionar el botón "Crear".
Pos-condiciones	
1.	Muestra en el navegador de Proyectos de Modelado el Proyecto de Modelado creado.
Flujos alternativos	
Flujo alternativo 1.A "Introducción del Nombre del Proyecto"	
1	Si el desarrollador no introduce el nombre del proyecto es por defecto "Nuevo Proyecto"
Pos-condiciones	
1	El nombre del proyecto de modelado es "Nuevo Proyecto"
Flujo alternativo 2.B "Introducción de la Notación de Modelado"	
1	Si el desarrollador no introduce la notación de modelado es por defecto "UML"
Pos-condiciones	
1	La notación de Modelado del proyecto de modelado es "UML"
Flujo alternativo 3.C "Introducción del Lenguaje de Programación"	
1	Si el desarrollador no introduce el Lenguaje de Programación es por defecto "Python"
Pos-condiciones	
1	El Lenguaje de Programación del proyecto de modelado es "Python"
Flujo alternativo 4.D "Introducción del Desarrollador"	
1	Si el desarrollador no introduce el Nombre del Desarrollador será por defecto el usuario conectado actualmente en la suite.
Pos-condiciones	
1	El nombre del Desarrollador del proyecto de modelado es el usuario conectado actualmente en la suite.
Flujo alternativo 5.E "Introducción de la Compañía"	
1	Si el desarrollador no introduce la nombre de la Compañía es por defecto en Blanco
Pos-condiciones	
1	No se registra ninguna Compañía.
Flujo alternativo 6.F "Introducción del Tipo de Software"	
1	Si el desarrollador no introduce el Tipo de Software es por defecto "Libre"
Pos-condiciones	
1	El Tipo de Software del proyecto de modelado es "Libre"
Flujo alternativo 7.G "Introducción de la Descripción"	
1	Si el desarrollador no introduce la descripción es por defecto en Blanco
Pos-condiciones	
1	La Descripción del proyecto de modelado es en Blanco
Validaciones	
1	Nombre del Proyecto
2	Notación de modelado
3	Lenguaje de programación
4	Desarrollador
5	Compañía
6	Tipo de software
7	Descripción del proyecto

Conceptos	Nombre del Proyecto:	Componente "Line Edit", no puede estar en blanco, no permite caracteres especiales.
	Notación de modelado:	Componente "Combo Box" (UML, BPMN)
	Lenguaje de programación:	Componente "Combo Box"
	Desarrollador:	Componente "Line Edit", no permite caracteres especiales.
	Compañía:	Componente "Line Edit", no permite caracteres especiales.
	Tipo de software:	Componente "Line Edit", no permite caracteres especiales.
	Descripción del proyecto:	Componente "Text Edit", no permite caracteres especiales.
Sección 2: "Cargar Proyecto de Modelado desde Rational Rose"		
Flujo Básico		
1.	Selecciona la opción "Cargar Proyecto de Modelado desde Rational Rose" en la siguiente ruta de la barra de menú, "Módulos/Modelado/Comenzar Proyecto de Modelado".	
2.	Introducir los siguientes datos Nombre del Proyecto, Notación de modelado, Lenguaje de programación, Desarrollador, Compañía, Tipo de software y Descripción del proyecto.	
3.	Seleccionar el archivo de Rational Rose a cargar que se encuentra salvado en la estación de trabajo en la que se encuentra el usuario.	
4.	Presionar el botón "Crear"	
Pos-condiciones		
1.	Muestra en el navegador de Proyectos de Modelado el Proyecto de Modelado cargado con los diagramas que posee.	
Flujos alternativos		
Flujo alternativo 1.A "Introducción del Nombre del Proyecto"		
1	Si el desarrollador no introduce el nombre del proyecto se por defecto "Nuevo Proyecto"	
Pos-condiciones		
1	El nombre del proyecto de modelado se "Nuevo Proyecto"	
Flujo alternativo 2.B "Introducción de la Notación de Modelado"		
1	Si el desarrollador no introduce la notación de modelado es por defecto "UML"	
Pos-condiciones		
1	La notación de Modelado del proyecto de modelado es "UML"	
Flujo alternativo 3.C "Introducción del Lenguaje de Programación"		
1	Si el desarrollador no introduce el Lenguaje de Programación es por defecto "Python"	
Pos-condiciones		
1	El Lenguaje de Programación del proyecto de modelado es "Python"	
Validaciones		
1	Nombre del Proyecto	
2	Notación de modelado	
3	Lenguaje de programación	
4	Desarrollador	
5	Compañía	
6	Tipo de software	
7	Descripción del proyecto	
Conceptos	Nombre del Proyecto:	Componente "Line Edit", no puede estar en blanco, no permite caracteres especiales.
	Notación de modelado:	Componente "Combo Box" (UML, BPMN)
	Lenguaje de programación:	Componente "Combo Box"
	Desarrollador:	Componente "Line Edit", no permite caracteres especiales.
	Compañía:	Componente "Line Edit", no permite caracteres especiales.
	Tipo de software:	Componente "Line Edit", no permite caracteres especiales.
	Descripción del proyecto:	Componente "Text Edit", no permite caracteres especiales.
Flujo alternativo 4.D "Introducción del Desarrollador"		

1	Si el desarrollador no introduce el Nombre del Desarrollador es por defecto el usuario conectado actualmente en la suite.	
Pos-condiciones		
1	El nombre del Desarrollador del proyecto de modelado es el usuario conectado actualmente en la suite.	
Flujo alternativo 5.E "Introducción de la Compañía"		
1	Si el desarrollador no introduce el nombre de la Compañía es por defecto en Blanco	
Pos-condiciones		
1	No se registra ninguna Compañía.	
Flujo alternativo 6.F "Introducción del Tipo de Software"		
1	Si el desarrollador no introduce el Tipo de Software es por defecto "Libre"	
Pos-condiciones		
1	El Tipo de Software del proyecto de modelado es "Libre"	
Flujo alternativo 7.G "Introducción de la Descripción"		
1	Si el desarrollador no introduce la descripción es por defecto en Blanco	
Pos-condiciones		
1	La Descripción del proyecto de modelado es en Blanco	
Flujo alternativo 8.H "Archivo de Rational Rose a cargar"		
1	Si el desarrollador no introduce un archivo con formato valido se le muestra un mensaje de error "Formato de archivo no válido".	
Pos-condiciones		
1	Mostrará un mensaje de error al desarrollador informándole que el formato del archivo no es correcto.	
Validaciones		
1	Nombre del Proyecto	
2	Notación de modelado	
3	Lenguaje de programación	
4	Desarrollador	
5	Compañía	
6	Tipo de software	
7	Descripción del proyecto	
Conceptos	Nombre del Proyecto:	Componente "Line Edit", no puede estar en blanco, no permite caracteres especiales.
	Notación de modelado:	Componente "Combo Box" (UML, BPMN)
	Lenguaje de programación:	Componente "Combo Box"
	Desarrollador:	Componente "Line Edit", no permite caracteres especiales.
	Compañía:	Componente "Line Edit", no permite caracteres especiales.
	Tipo de software:	Componente "Line Edit", no permite caracteres especiales.
	Descripción del proyecto:	Componente "Text Edit", no permite caracteres especiales.
Sección 3: "Cargar Proyecto de Modelado desde formato XMI"		
Flujo Básico		
1	Selecciona la opción "Cargar Proyecto de Modelado desde formato XMI" en la siguiente ruta de la barra de menú, "Módulos/Modelado/Comenzar Proyecto de Modelado".	
2	Introducir los siguientes datos Nombre del Proyecto, Notación de modelado, Lenguaje de programación, Desarrollador, Compañía, Tipo de software y Descripción del proyecto.	
3	Seleccionar el archivo en formato XMI a cargar que se encuentra salvado en la máquina donde se está trabajando.	
4	Presionar el botón "Crear"	
Pos-condiciones		
1.	Muestra en el navegador de Proyectos de Modelado el Proyecto de Modelado cargado con los diagramas que posee.	
Flujos alternativos		
Flujo alternativo 1.A "Introducción del Nombre del Proyecto"		
1	Si el desarrollador no introduce el nombre del proyecto es por defecto "Nuevo Proyecto"	
Pos-condiciones		
1	El nombre del proyecto de modelado es "Nuevo Proyecto"	
Flujo alternativo 2.B "Introducción de la Notación de Modelado"		

1	Si el desarrollador no introduce la notación de modelado es por defecto "UML"	
Pos-condiciones		
1	La notación de Modelado del proyecto de modelado es "UML"	
Flujo alternativo 3.C "Introducción del Lenguaje de Programación"		
1	Si el desarrollador no introduce el Lenguaje de Programación es por defecto "Python"	
Pos-condiciones		
1	El Lenguaje de Programación del proyecto de modelado es "Python"	
Flujo alternativo 4.D "Introducción del Desarrollador"		
1	Si el desarrollador no introduce el Nombre del Desarrollador es por defecto el usuario conectado actualmente en la suite.	
Pos-condiciones		
1	El nombre del Desarrollador del proyecto de modelado es el usuario conectado actualmente en la suite.	
Flujo alternativo 5.E "Introducción de la Compañía"		
1	Si el desarrollador no introduce la nombre de la Compañía es por defecto en Blanco	
Pos-condiciones		
1	No se registra ninguna Compañía.	
Flujo alternativo 6.F "Introducción del Tipo de Software"		
1	Si el desarrollador no introduce el Tipo de Software es por defecto "Libre"	
Pos-condiciones		
1	El Tipo de Software del proyecto de modelado es "Libre"	
Flujo alternativo 7.G "Introducción de la Descripción"		
1	Si el desarrollador no introduce la descripción es por defecto en Blanco	
Pos-condiciones		
1	La Descripción del proyecto de modelado es en Blanco	
Flujo alternativo 8.H "Archivo de XMI a cargar"		
1	Si el desarrollador no introduce un archivo con formato valido se le muestra un mensaje de error "Formato de archivo no válido".	
Pos-condiciones		
1	Muestra un mensaje de error al desarrollador informándole que el formato del archivo no es correcto.	
Validaciones		
1	Nombre del Proyecto	
2	Notación de modelado	
3	Lenguaje de programación	
4	Desarrollador	
5	Compañía	
6	Tipo de software	
7	Descripción del proyecto	
Conceptos	Nombre del Proyecto:	Componente "Line Edit", no puede estar en blanco, no permite caracteres especiales.
	Notación de modelado:	Componente "Combo Box" (UML, BPMN)
	Lenguaje de programación:	Componente "Combo Box"
	Desarrollador:	Componente "Line Edit", no permite caracteres especiales.
	Compañía:	Componente "Line Edit", no permite caracteres especiales.
	Tipo de software:	Componente "Line Edit", no permite caracteres especiales.
	Descripción del proyecto:	Componente "Text Edit", no permite caracteres especiales.
Requisitos especiales	No procede	
Asuntos pendientes	No procede	

Tabla 12: Descripción del requisito funcional Comenzar Proyecto de Modelado.

Prototipo elemental de interfaz gráfica de usuario. Comenzar Proyecto de Modelado.

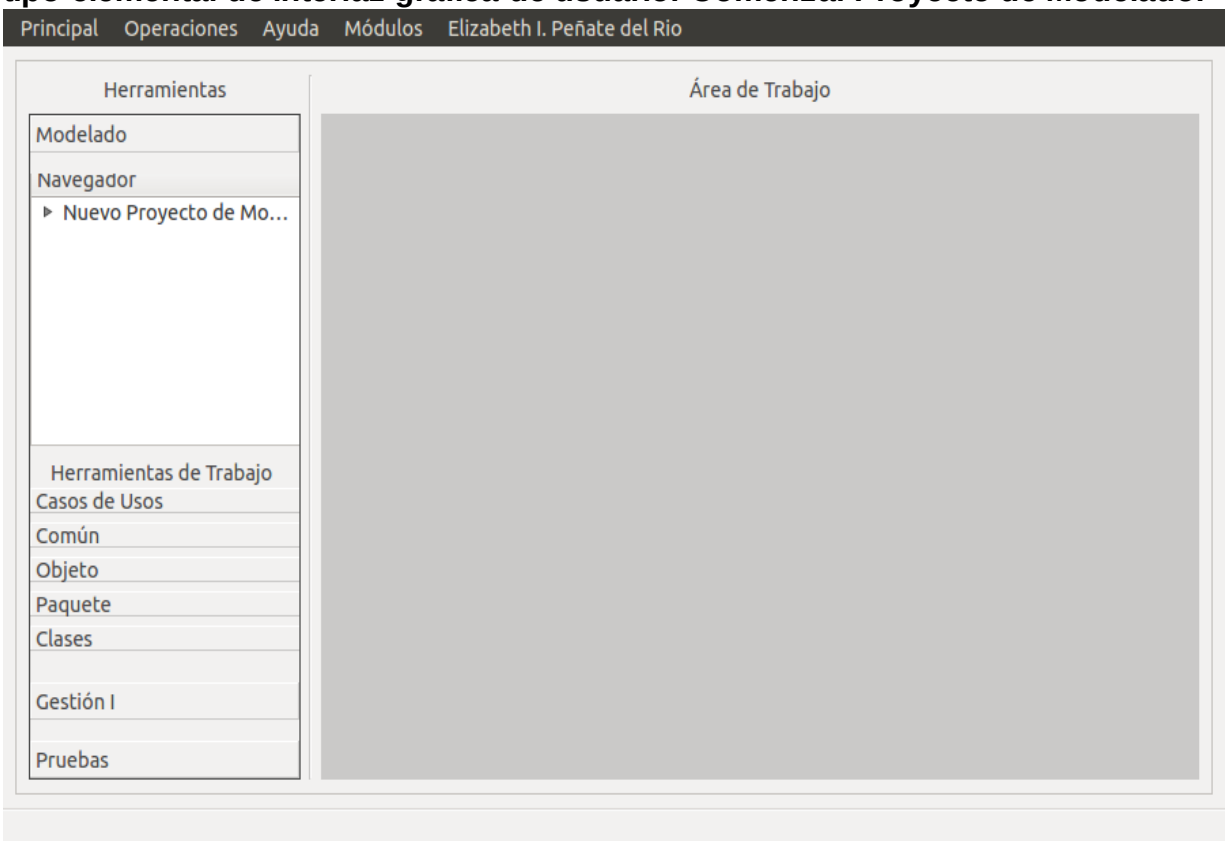

Figura 1: Interfaz Principal

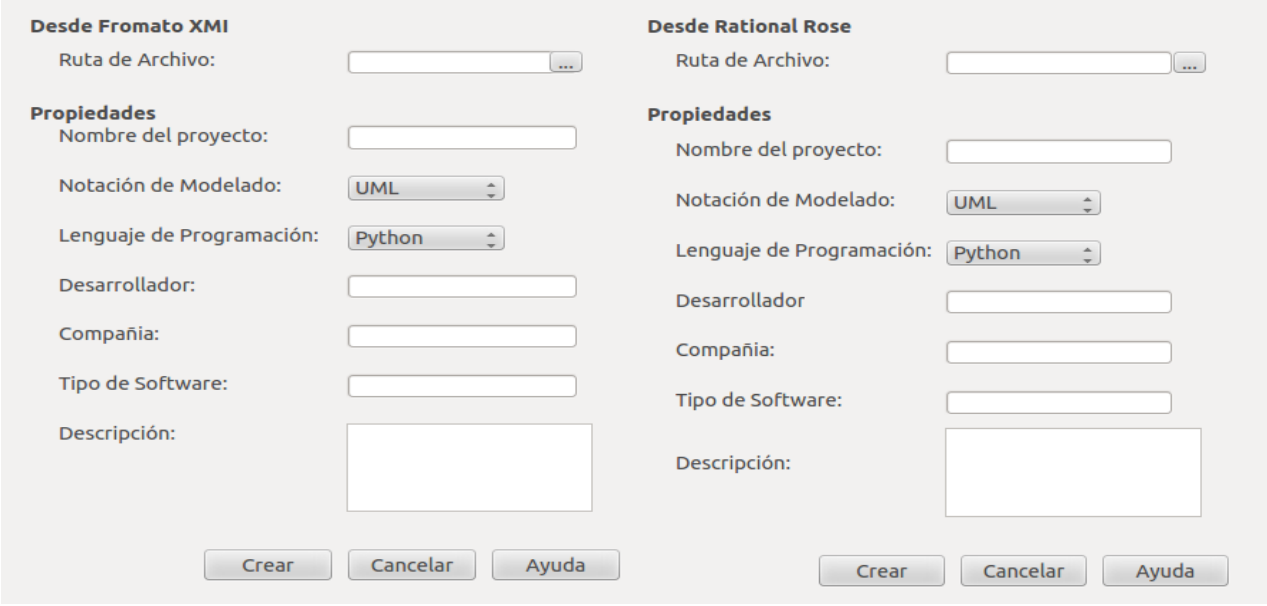

Figura 2: Interfaces Comenzar Proyecto de Modelado

Para más detalles ver Plantilla 0129_Descripción de requisitos.doc del Nova-QALIT.

3. Propuesta de la Arquitectura

La arquitectura a utilizar sigue el patrón 3 Capas que se encuentra dentro de la clasificación de los estilos de llamada y retorno. Específicamente va a contar con 3 capas: Presentación, Lógica del Negocio y Acceso a Datos.

Capa de Presentación: En esta capa se encuentran las clases de las interfaces que interactúan con el usuario. Esta es la mediadora entre la capa de Lógica del Negocio y el usuario, mostrando la respuesta que le proporcione la capa inmediatamente inferior, Lógica del negocio a las peticiones del usuario. Esta capa también contiene las librerías que facilitan el dibujo de los diagramas, teniendo en cuenta los tipos de estereotipos.

Lógica del Negocio: En esta capa es donde recae el mayor peso de la aplicación ya que va a contener todas las clases manejadoras que controlan el desarrollo de la mayoría de las actividades apoyándose en las demás clases para lograr obtener los diagramas requeridos y finalmente usarlos evidenciando así la relación con la Capa de Acceso a Datos. Esta capa también va a contener un paquete de configuración para determinar la configuración específica de estas clases.

Capa de Acceso a Datos: La capa de Acceso a Datos recibe las peticiones de la capa inmediatamente superior, Lógica del Negocio, realizados por el desarrollador, esta contiene todas las clases entidades donde se recogen los datos importantes para el desarrollo del sistema.

4. Vistas Arquitectónicas del Módulo de Modelado

4.1. Vista de Casos de Uso

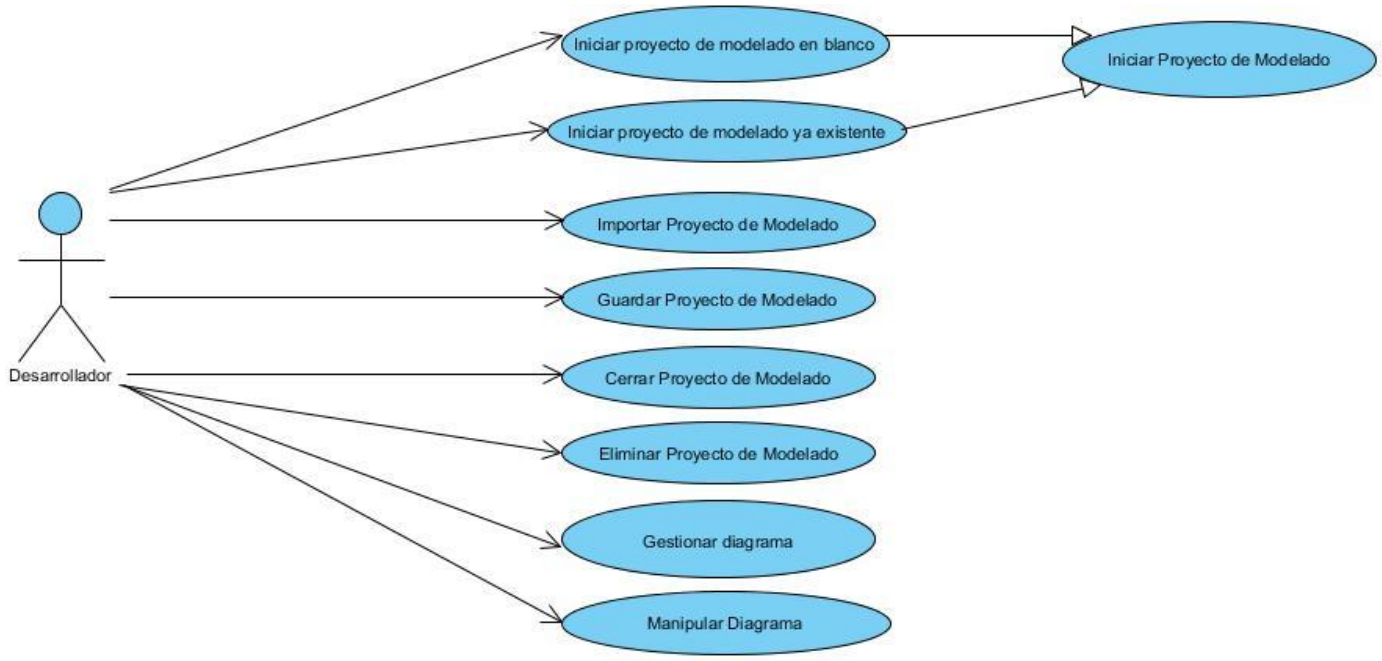

Figura 3: Diagrama de CU del Sistema

Este diagrama muestra el comportamiento de los actores y casos de uso del sistema. El desarrollador es el único actor del sistema, quien inicializa todos los casos de uso del mismo.

Descripción del actor: Desarrollador: Este tiene como meta hacer los modelos necesarios para definir un producto software teniendo en cuenta las diferentes vistas arquitectónicas.

4.2 Vista Lógica

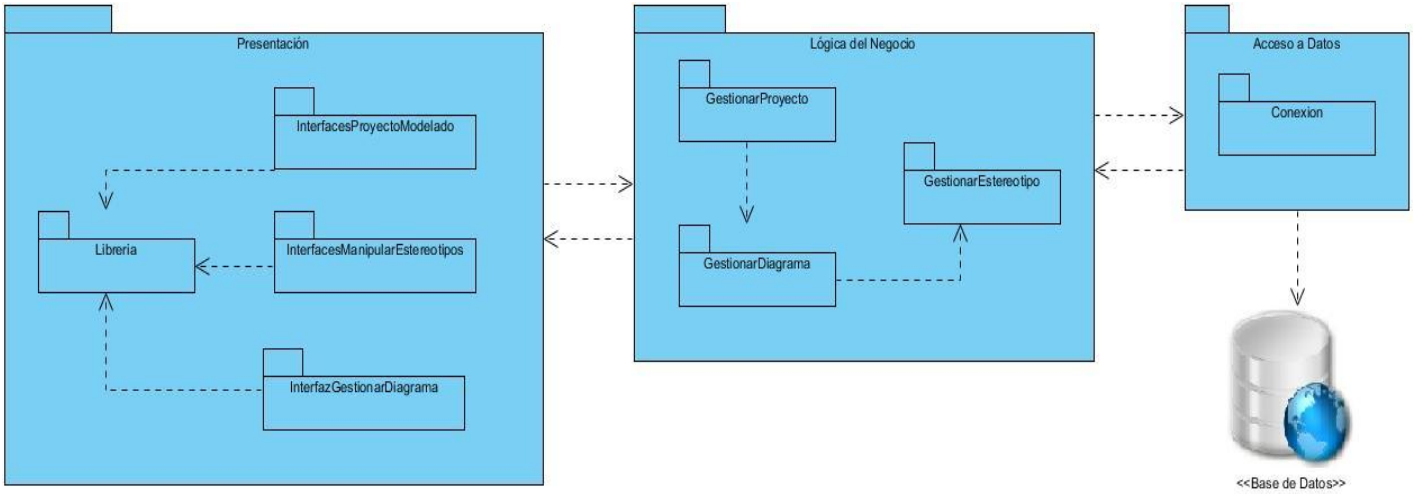

Figura 4: Diagrama de Paquetes

Los principales paquetes por los que está conformado el sistema son:

Capa de Presentación.

Paquete Librería: Donde se encuentran las librerías que tienen que ver con las interfaces del sistema.

Paquete InterfacesProyectoModelado: Es el que agrupa todas las interfaces que tienen que ver con el proyecto de modelado.

Paquete InterfacesManipularEstereotipo: Agrupa todas las interfaces que tienen que ver con la gestión de los estereotipos.

Paquete InterfacesGestionarDiagrama: Es la que agrupa todas las interfaces que tienen que ver con la gestión de los diagramas.

Capa Lógica del Negocio.

Paquete GestionarProyecto: Es donde se encuentra la clase encargada de controlar todo lo relacionado al proyecto de modelado.

Paquete GestionarDiagrama: Es donde se encuentra la clase encargada de controlar todo lo relacionado con los diagramas.

Paquete GestionarEstereotipo: Es donde se encuentra la clase encargada de controlar todo lo relacionado con los estereotipos.

Capa de Acceso a Datos.

Paquete Conexión: Es donde se encuentra la clase encargada de establecer la conexión con la base de datos.

Base de datos: Es donde se encuentran todas las tablas y sus relaciones según las entidades del sistema.

4.3 Vista de Implementación

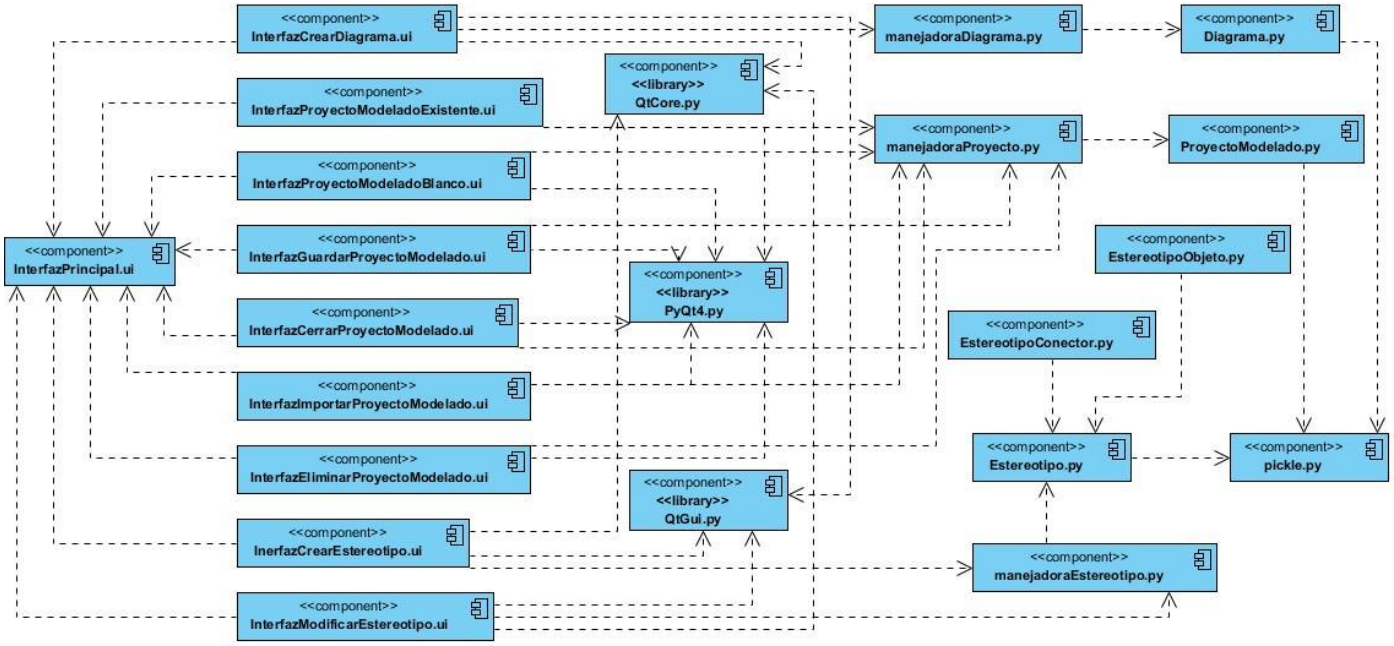

Figura 5: Diagrama de Componentes.

Esta vista está conformada por los siguientes componentes:

InterfazPrincipal.ui: Es la que contiene el área de trabajo y las herramientas a utilizar en la misma.

InterfazCrearDiagrama.ui: Es la encargada según el tipo de diagrama de crearlo.

InterfazProyectoModeladoExistente.ui: Es la encargada de crear un proyecto de modelado utilizando uno ya existente en otro formato.

InterfazProyectoModeladoBlanco.ui: Es a encargada de crear un proyecto de modelado en blanco.

InterfazGuardarProyectoModelado.ui: Es la encargada de guardar un proyecto de modelado según el formato que se desee.

InterfazCerrarProyectoModelado.ui: Es la encargada de cerrar un proyecto de modelado.

InterfazImportarProyectoModelado.ui: Es la encargada de importar un proyecto de modelado según el formato que desee.

InterfazEliminarProyectoModelado.ui: Es la encargada de eliminar un proyecto de modelado existente.

InterfazCrearEstereotipo.ui: Es la encargada de crear un estereotipo, según el tipo de estereotipo que se desee.

InterfazModificarEstereotipo.ui: Es la encargada de modificar el estereotipo que se desee.

QtCore.py: Es la librería encargada del color y el tamaño de los dibujos.

PyQt4.py: Es la librería encargada de crear las interfaces y generarlas a código Python.

QTGui.py: Es la librería encargada de dibujar los diagramas.

manejadoraDiagrama.py: Es la encargada de controlar todas las funcionalidades de los diagramas.

manejadoraProyecto.py: Es la encargada de controlar todas las funcionalidades del Proyecto de Modelado.

manejadoraEstereotipo.py: Es la encargada de controlar todas las funcionalidades de los estereotipos.

Pickle.py: Es la clase encargada de la conexión con la base de datos.

Estos componentes están contenidos en la base de datos:

Diagrama.py: Es la que contiene toda la información necesaria de los diagramas.

ProyectoModelado.py: Es la que contiene toda la información de los proyectos de modelado.

Estereotipo.py: Es la que contiene toda la información general de los estereotipos.

EstereotipoObjeto.py: Es la que contiene toda la información relacionada con los estereotipos objetos.

EstereotipoConector.py: Es la que contiene toda la información relacionada con los estereotipos conectores.

Diagrama de Clases del Diseño

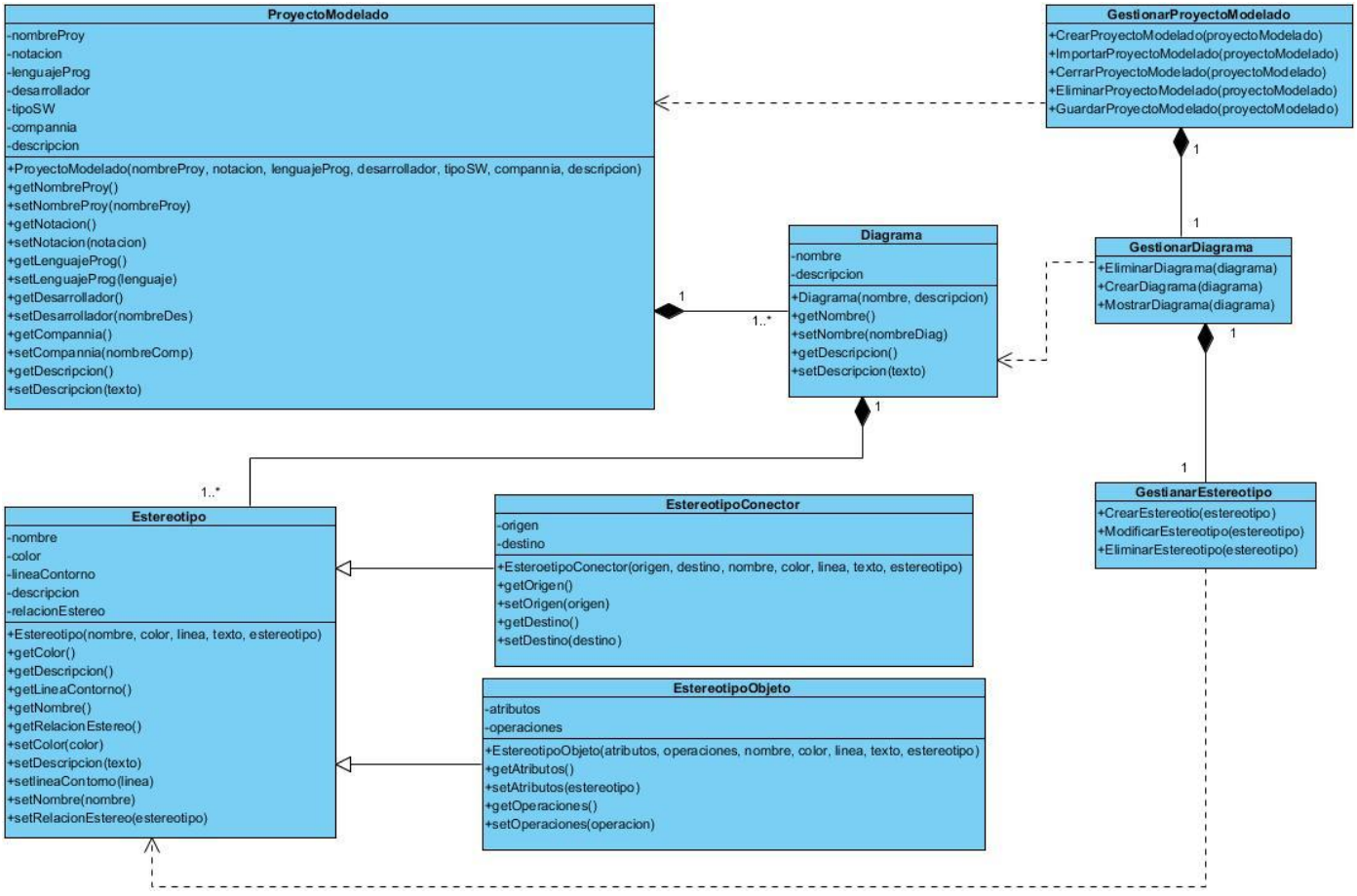

Figura 6: Diagrama de clases del diseño

Este diagrama consta de 8 clases:

Clase ProyectoModelado: Que agrupa todos los atributos que posee un proyecto de modelado, estos son nombreProy, notacion, lenguajeProg, desarrollador, tipoSW, compannia, descripcion todos con tipo de dato string.

Clase GestionarProyectoModelado: Que agrupa todos los métodos de un proyecto de modelado, controlando todas las funcionalidades que se realicen.

Clase Diagrama: Agrupa todos los atributos que posee un diagrama, estos son nombre, descripcion con tipo de dato string.

Clase GestionarDiagrama: Agrupa todos los métodos a utilizar en el diseño de un diagrama.

Clase Estereotipo: Es la que agrupa todos los atributos que posee un estereotipo, estos son nombre, color, lineaContorno, descripcion todos tipo de dato string y relacionEstereotipo tipo de dato estereotipo.

Clase EstereotipoConector: Contiene todos los atributos que posee un conector, estos son origen y destino, con tipo de dato estereotipo.

Clase EstereotipoObjeto: Contiene todos los atributos que posee un objeto, estos son atributos y operaciones, con tipo de dato array.

Clase GestionarEstereotipo: Es la encargada de controlar todas las funcionalidades de los estereotipos, según el tipo que sea.

En este diagrama se ponen de manifiesto varios patrones de diseño como son el experto que se evidencia en todas las clases puesto que cada una de ellas cuenta con la información necesaria para realizar sus responsabilidades. Por ejemplo, la clase GestionarProyectoModelado contiene toda la información que necesita para crear, importar, cerrar, eliminar y guardar un proyecto de modelado, igual así con las demás clases. También se evidencia el patrón bajo acoplamiento, puesto que como todas las clases son expertas y cada una según la información que contiene realiza sus responsabilidades, existe poca dependencia entre las clases y esto ayuda a que el tiempo de ejecución del sistema no sea grande. Evidenciándose también el patrón alta cohesión, permitiendo que las clases coexistan entre sí, es decir, que según las características que tenga la clase es que realiza las responsabilidades. Por ejemplo la clase estereotipo, que contiene todo lo relacionado a los estereotipos es la que puede manipularlos. Otro patrón de diseño aplicado es el creador, el cual guía la asignación de responsabilidades a la hora de crear instancias de otras clases. Por ejemplo la clase estereotipo es la única que puede crear una instancia de sus clases hijas estereotipoConector y estereotipoObjeto.

4.4 Vista de despliegue

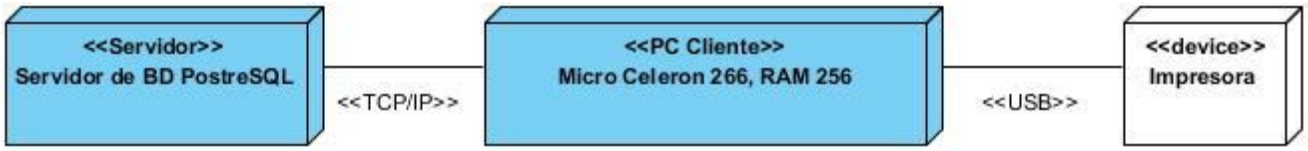

Figura 7: Diagrama de despliegue.

La vista de despliegue que se muestra es la general de la suite la cual está compuesta por diferentes módulos, estos utilizan los diagramas que se generan en el módulo de modelado para generar artefactos o reportes del sistema, los cuales se pueden imprimir.

Servidor de Base de Datos: Es donde se van a almacenar todos los datos del sistema. Donde se va a almacenar e integrar toda la información que se procesa por el resto de los módulos.

PC Cliente: Son las estaciones de trabajo a utilizar por los usuarios para acceder a la aplicación.

Impresora: Dispositivo que va a realizar el proceso de impresión de la documentación generada y los diagramas que necesiten ser impresos.

TCP/IP: Protocolo que representa la manera en la que se realizan las comunicaciones entre el servidor de Bases de Datos con la PC Cliente.

USB: Protocolo que permite adjuntar los dispositivos periféricos como la impresora a la pc cliente.

Conclusiones parciales

En este capítulo para la estructuración de la arquitectura se usa el modelo 4 vistas + 1 para una mayor organización del sistema, definiendo las vistas arquitectónicas del módulo de modelado. Se determinan los patrones arquitectónicos y de diseño, 3 capa, Creador, Experto, Bajo acoplamiento y Alta cohesión, para llevar a cabo una arquitectura robusta. Garantizando así la representación de los modelos que forman parte de la arquitectura de software.

Capítulo #3: Evaluación de la arquitectura de la herramienta de modelado para la Suite de Ingeniería de Software

Introducción

En este capítulo se presentan los resultados al evaluar la arquitectura de la herramienta de modelado para determinar el cumplimiento de los requisitos y gestionar los riesgos que pueda correr la misma, de modo que se obtenga una arquitectura robusta. Para esto se tienen en cuenta los métodos para la evaluación de la arquitectura seleccionándose el más adecuado. Utilizando diferentes pasos, como determinar los atributos de calidad del sistema, realizar una priorización y ordenamiento de los escenarios determinados según el método de evaluación y luego organizarlos en un árbol de utilidad. Identificando así los posibles riesgos, tomándose decisiones para la corrección de los mismos.

1. Atributos de calidad a evaluar en la arquitectura

1.1. Modelo Boehm

Este modelo define a la calidad de software mediante un conjunto de atributos organizados jerárquicamente, los criterios de calidad se presentan en tres grandes subdivisiones. La primera división es hecha acorde a los servicios que el sistema va a ofrecer (portabilidad). La segunda se hace de acuerdo a la operación del producto (usabilidad) y la tercera gran subdivisión se hace de acuerdo a la mantenibilidad del producto de software.

Entre los criterios básicos del modelo podemos mencionar:

- Usabilidad. Este atributo de calidad de software se enfoca a mejorar la simplicidad, entendimiento y facilidad de uso de un sistema de software para un cliente o usuario final.
- Mantenibilidad. Tradicionalmente de define como el esfuerzo requerido para localizar y especificar un error en la operación de un módulo, función o sistema de software.
- Portabilidad. Se define como el esfuerzo requerido para transportar la configuración de hardware y/o software un módulo, función o sistema de software en el ambiente de una plataforma a otra.
- Confiabilidad. Es una propiedad que implica el grado de confianza esperado por parte del usuario en la operación adecuada del sistema al utilizarlo. La confiabilidad es afectada por cuatro aspectos fundamentales:
  - Disponibilidad. Define la probabilidad de que el sistema esté funcionando en un tiempo determinado.
  - Fiabilidad. Es la probabilidad de que el sistema funcione correctamente durante un intervalo de tiempo específico.
  - Seguridad. Representa la capacidad de que el sistema no afecte su entorno y el de quien lo utiliza.

- Protección. Representa la capacidad del sistema para protegerse de intrusiones accidentales o programadas.

- Eficiencia. Se refiere al esfuerzo en base al costo de los recursos de la computadora que emplea un sistema, módulo o función de software para su ejecución.
- Ingeniería social. Este atributo se refiere al entendimiento y aceptación del sistema de software del grupo social al cual se enfocaron los requerimientos del sistema.
- Pruebas. Es el conjunto de exámenes que se realizan a un sistema, módulo o función una vez finalizado su desarrollo. Estos exámenes se realizan con el objetivo de asegurar que el sistema, módulo o función, cumple con la especificación de los requerimientos del usuario o cliente final.
- Entendibilidad. Es la claridad en cuanto a la lógica del código fuente de un módulo, función o sistema de software.
- Modificabilidad. Este atributo está muy ligado a la Entendibildad, debido a que si un código es entendible, como consecuencia podrá ser cambiado en su contenido sin que esto afecte negativamente al sistema en el cual se integra.
- Flexibilidad. Representa el esfuerzo requerido para modificar el código de un sistema, módulo o función de software en operación.
- Reusabilidad. El alcance para el cual un programa puede ser usado en otras aplicaciones.
- Interoperatividad. Se define como el esfuerzo requerido para acoplar un sistema a otro. [42]

1.2. Modelo McCall

La calidad se define de forma jerárquica y resuelve la complejidad mediante la descomposición. Realiza la revisión en el producto y la calidad en el proceso y en la operación del producto. Se focaliza en el producto final, identificando atributos claves desde el punto de vista del usuario.

Este modelo establece tres áreas principales que intervienen en la Calidad del Software:

- Calidad en la operación del producto. En general se requiere que este pueda ser entendido fácil, que opere eficientemente y que los resultados sean los requeridos inicialmente por el usuario.
- Revisión de calidad del producto. Tiene como objetivo realizar revisiones durante el proceso de desarrollo para detectar los errores que afecten a la operación del producto de software. Las revisiones involucran grupos de personas que examinan parte o todo el proceso del software, los sistemas o su documentación asociada para descubrir problemas potenciales. Las conclusiones de la revisión se registran formalmente y se pasan al autor o a quien sea responsable de corregir los problemas descubiertos.
- Calidad en el proceso. Desde este enfoque se recomienda definir o seleccionar estándares y procedimiento que sirvan como marco de trabajo durante el desarrollo de software. En la figura 2.2 se pueden apreciar la relación que guardan estos tres aspectos. [42]

1.3. Modelo ISO 9126

El estándar ISO-9126 establece que cualquier componente de la calidad del software puede ser descrito en términos de una o más de seis características básicas, las cuales son: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portatilidad; cada una de las cuales se detalla a través de un conjunto de subcaracterísticas que permiten profundizar en la evaluación de la calidad de productos de software.

Características:

Funcionalidad: En este grupo se agrupan una serie de atributos que permiten calificar si un producto de software maneja en forma adecuada el conjunto de funciones que satisfagan las necesidades para las cuales fue diseñado. Para este propósito se establecen los siguientes atributos:

- Idoneidad: Es la capacidad del software para mantener un conjunto apropiado de funciones para las tareas y objetivos especificados por los usuarios.
- Precisión: Capacidad del software para proporcionar efectos o resultados correctos o convenidos con el grado de exactitud necesario.
- Seguridad: Capacidad del producto de software para proteger la información y los datos, para que personas o sistemas desautorizados no puedan leer o modificar los mismos, y las personas o sistemas autorizados tenga el acceso a ellos.
- Interoperabilidad: Capacidad del producto de software para interactuar recíprocamente con uno o más sistemas especificados.
- Conformidad con la funcionalidad: Capacidad del software de apegarse a la normas, leyes, regulaciones y similares que se le apliquen en términos relativos a la funcionalidad.

Confiabilidad: Aquí se agrupan un conjunto de atributos que se refieren a la capacidad del software de mantener su nivel de ejecución bajo condiciones normales en un periodo de tiempo establecido. Las sub-características que el estándar sugiere son:

- Madurez: Capacidad del producto de software de evitar un fallo total como resultado de producirse un fallo del software.
- Tolerancia ante fallos: Capacidad del software de mantener un nivel de ejecución o desempeño especificado en caso de fallos del software.
- Recuperabilidad: Capacidad del producto de software de restablecer un nivel de ejecuciones especificado y recuperar los datos directamente afectados en caso de fallo total.
- Conformidad con la confiabilidad: Capacidad del producto de software para adherirse a las normas que se le apliquen, convenciones, regulaciones, leyes y las prescripciones similares relativas a la confiabilidad.

Usabilidad: Consiste de un conjunto de atributos que permiten evaluar el esfuerzo necesario que deberá invertir el usuario para utilizar el sistema.

- Comprensibilidad: capacidad del producto de software para permitirle al usuario entender si el software es idóneo, y cómo puede usarse para las tareas y condiciones de uso particulares.
- Cognoscibilidad: capacidad del producto de software para permitirle al usuario aprender su aplicación.
- Operabilidad: capacidad del producto de software para permitirle al usuario operarlo y controlarlo.
- Atracción: capacidad del producto de software de ser atractivo o amigable para el usuario.
- Conformidad con la usabilidad: capacidad del producto de software para adherirse a las normas, convenciones, guías de estilo o regulaciones relativas a la usabilidad.

Eficiencia: Esta característica permite evaluar la relación entre el nivel de funcionamiento del software y la cantidad de recursos usados. Los aspectos a evaluar son:

- Rendimiento: capacidad del producto de software para proporcionar apropiados tiempos de respuesta y procesamiento, así como tasas de producción de resultados al realizar su función bajo condiciones establecidas.
- Utilización de recursos: capacidad del producto de software para utilizar la cantidad y el tipo apropiado de recursos cuando realiza su función bajo las condiciones establecidas.
- Conformidad de la eficiencia: capacidad del producto de software de adherirse a las normas o convenciones que se relacionan con la eficiencia.

Mantenibilidad: Se refiere a los atributos que permiten medir el esfuerzo necesario para realizar modificaciones al software, ya sea por la corrección de errores o por el incremento de funcionalidad. En este caso, se tienen los siguientes factores:

- Diagnosticabilidad: capacidad del producto de software de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.
- Flexibilidad: capacidad del producto de software para permitir la aplicación de una modificación especificada.
- Estabilidad: capacidad del producto de software para minimizar los efectos inesperados de las modificaciones realizadas.
- Contrastabilidad: capacidad del producto del software para permitir la validación de modificaciones realizadas a su código.
- Conformidad de la mantenibilidad: capacidad del producto de software para adherirse a las normas o convenciones que se relacionan con la mantenibilidad.

Portatilidad: En este caso, se refiere a la habilidad del software de ser transferido de un ambiente a otro, y considera los siguientes aspectos:

- Adaptabilidad: capacidad del producto de software de ser adaptado a los ambientes especificados sin aplicar acciones o medios de otra manera que aquellos suministrados con el propósito de que el software cumpla sus fines.
- Instalabilidad: capacidad del producto de software de ser instalado en un ambiente especificado.
- Coexistencia: capacidad del producto de software de coexistir con otro software independiente en un ambiente común y compartir los recursos comunes.
- Remplazabilidad: capacidad del producto de software de ser usado en lugar de otro producto de software especificado para los mismos fines y en el mismo ambiente.
- Conformidad con la portabilidad: capacidad del producto de software de adherirse a las normas o convenciones relativas a la portabilidad. [43]

De los atributos de calidad estudiados para evaluar la arquitectura se utilizan los de la ISO 9126, puesto que está basada en los modelos antes mencionados, McCall y Boehm. Además provee una descomposición de las características en subcaracterísticas y se hace una distinción entre calidad interna, calidad externa y calidad en uso.

2 Métodos de evaluación de la arquitectura

La evaluación de la arquitectura, se utiliza para identificar riesgos potenciales en su estructura y en sus propiedades, que puedan afectar al sistema de software resultante. Verifica que los requisitos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Asegurando que el sistema a construir cumpla con las necesidades de los usuarios. Existen varios métodos que permiten realizar la evaluación de la Arquitectura de Software, a continuación se analiza el Método de Análisis de Desventajas de la Arquitectura (Architecture Trade-off Analysis Method, ATAM), el cual por las características que presenta se ajusta al presente trabajo.

2.1. Método de Análisis de Desventajas de Arquitectura (ATAM)

El Método de Análisis de Desventajas de Arquitectura (ATAM) es una metodología para evaluar Arquitecturas de Software que principalmente evalúa la adecuación de la Arquitectura de Software definida con respecto a los atributos de calidad especificados para el sistema. Surge construyendo ideas y técnicas de tres áreas: la noción de estilos o patrones de arquitectura, el análisis de atributos de calidad y el Método de Análisis de Arquitectura de Software (SAAM) que es el predecesor del ATAM.

ATAM brinda una caracterización para varios atributos creada a partir del conocimiento existente en las comunidades de atributos de calidad, principalmente para atributos como performance, modificabilidad, disponibilidad, usabilidad y seguridad. Cada caracterización de atributos de calidad se divide en tres

categorías: estímulo externo, decisiones arquitectónicas y respuestas, donde los primeros son eventos que hacen que la arquitectura responda o cambie, las decisiones arquitectónicas son los elementos de la arquitectura (componentes, conectores y sus propiedades) que tienen impacto directo en la obtención de respuestas a los atributos, y las respuestas se caracterizan por atributos medibles como latencia y throughput.

La primera tarea a realizar en una evaluación de arquitectura es obtener una especificación precisa de los objetivos de calidad contra los cuales la arquitectura será evaluada. El mecanismo utilizado para obtener esta información es el Escenario. Los escenarios proveen una forma de concretizar cualidades a tener en cuenta en tiempo de desarrollo como la modificabilidad, ayudan a entender cualidades de tiempo de ejecución como performance o disponibilidad, entre otros. Se definen tres tipos de escenarios, estos casos distintos de escenarios se utilizan para probar el sistema desde distintos puntos de vista, permitiendo que no se pasen por alto decisiones arquitectónicas que puedan representar riesgos.

La información de los escenarios es obtenida y priorizada en ATAM utilizando dos mecanismos diferentes en distintos momentos, y con la participación de diferentes stakeholders, estos son los árboles de utilidad y la lluvia de ideas estructurada. Los árboles de utilidad se realizan con el arquitecto y proveen un mecanismo de enfoque top-down para traducir los principales requerimientos del negocio para el sistema, en escenarios concretos de atributos de calidad.

ATAM consiste en la ejecución de nueve pasos que se dividen en cuatro grupos que a su vez, se realizan en el tiempo en cuatro Fases diferenciadas. Estos cuatro grupos no se corresponden uno a uno con las Fases, sino que se realizan en las Fases 1 y 2, y se agrega una Fase 0 previa de preparación y una Fase 3 posterior de finalización del proyecto.

En la Fase 0 se acuerdan tiempo, fechas, costos, esfuerzo y se forma el equipo de evaluación, en las Fases 1 y 2 se realiza la evaluación con ATAM siguiendo los nueve pasos especificados, y finalmente en la Fase 3 se realiza el informe final de la evaluación realizada, se recoge información para la medida y mejora del proceso y se actualizan los repositorios de productos generados.

Fase 1: En la Fase 1 se realizan las actividades correspondientes a los grupos de presentación e investigación y análisis.

El grupo de presentación se compone de tres pasos:

1 – Presentar el ATAM.

2 – Presentar las pautas del negocio.

3 – Presentar la Arquitectura.

El segundo grupo de investigación y análisis se compone también de tres pasos:

4 – Identificar las propuestas arquitectónicas.

5 – Generar el árbol de utilidad de los atributos de calidad.

6 – Analizar las propuestas arquitectónicas.

Fase 2: En la Fase 2 se realizan las actividades incluidas en el tercer grupo de pruebas y el cuarto grupo de informes, que completan los tres pasos restantes.

El grupo de pruebas se compone de dos pasos:

7 – Lluvia de ideas.

8 – Analizar las propuestas arquitectónicas.

El grupo de Informes se compone de un solo paso:

9– Presentar los resultados. [44]

3 Aplicación del método escogido a la arquitectura.

Para aplicar este método se tiene en cuenta los siguientes pasos.

3.1 Atributos a medir según ISO 9126

Los atributos a medir según la ISO 9126 son los abordados en el epígrafe 1 Atributos de calidad a evaluar en la arquitectura, específicamente el epígrafe 1.3 ISO 9126.

Funcionabilidad

- Idoneidad
- Precisión.
- Seguridad.
- Interoperabilidad.
- Conformidad con la funcionalidad.

Confiabilidad

- Madurez.
- Tolerancia ante fallos.
- Recuperabilidad.
- Conformidad con la confiabilidad.

Usabilidad

- Comprensibilidad.
- Cognoscibilidad.
- Operabilidad.
- Atracción.
- Conformidad con la usabilidad.

Eficiencia

- Rendimiento.
- Utilización de recursos.
- Conformidad de la eficiencia.

Mantenibilidad

- Diagnosticabilidad.
- Flexibilidad.
- Estabilidad.
- Conformidad de la mantenibilidad.

Portabilidad

- Adaptabilidad.
- Instalabilidad.
- Coexistencia.
- Remplazabilidad.
- Conformidad con la portabilidad.

3.2 Priorización y Ordenamiento de escenarios.

Escenario 1: El producto propiciará aceptables tiempos de respuesta y procesamiento de la información.
Atributos: **Eficiencia - Rendimiento.**
Estímulo: Guardar, importar, exportar, cerrar y eliminar los proyectos de modelado y cambios realizados en dichos proyectos.
Contexto: Se guarda, importa, exporta, cierra o eliminan los proyectos de modelado.
Respuesta: Muestra una ventana de confirmación para guardar los cambios realizados a la hora de guardar o cerrar un proyecto de modelado y a la hora de eliminar un proyecto de modelado.

Escenario 2: El producto permite que se le realicen diferentes modificaciones específicas.
Atributos: **Mantenibilidad - Flexibilidad.**
Estímulo: Realizar cambios en el diseño de los diagramas y estereotipos en dependencia del tipo.
Contexto: Se realiza el cambio especificado.
Respuesta: Los cambios realizados por el desarrollador son aceptados por la aplicación.

Escenario 3: El producto tiene la capacidad de minimizar los efectos ante cambios que se realicen.
Atributos: **Mantenibilidad - Estabilidad.**
Estímulo: Realizar modificaciones en los diagramas creados.
Contexto: Se realiza el cambio específico.
Respuesta: Se muestran los cambios realizados.

Escenario 4: El producto presenta gran capacidad para la protección de la información y los datos.
Atributos: **Funcionabilidad - Seguridad.**
Estímulo: Guardar cambios realizados en el proyecto de modelado.

Contexto: Se guarda el cambio realizado al instante.

Respuesta: Los cambios son guardados en la base de datos.

Escenario 5: El producto está capacitado para el mantener un nivel de ejecución ante cualquier fallo de software.

Atributos: **Confiabilidad - Tolerancia ante fallos.**

Estímulo: Exportar un diagrama.

Contexto: Exporta el diagrama especificado.

Respuesta: Muestra la interfaz para exportar un diagrama según el tipo especificado.

Escenario 6: El producto presenta características que permiten al usuario entender el software y realizar sus tareas de una manera más entendible.

Atributos: **Usabilidad - Comprensibilidad.**

Estímulo: Crear un nuevo diagrama.

Contexto: Crea el diagrama especificado.

Respuesta: Muestra la interfaz para la creación de un nuevo diagrama según el tipo especificado.

Escenario 7: El producto presenta capacidades para la adaptación a otros ambientes, tales como Windows y Linux.

Atributos: **Portabilidad - Adaptabilidad.**

Estímulo: Aplicable a entornos de Windows y Linux.

Respuesta: El producto trabaja sobre el sistema operativo Windows y Linux.

Escenario 8: Producto con funciones adecuadas que permite ser instalada en un ambiente específico.

Atributos: **Portabilidad - Instalabilidad.**

Estímulo: Aplicable a entornos de Windows y Linux.

Respuesta: El producto trabaja sobre el sistema operativo Windows y Linux.

Escenario 9: El producto de software posee la capacidad de coexistir con otro software independiente en un ambiente común y compartir los recursos comunes.

Atributo: **Portabilidad - Coexistencia.**

Estímulo: Inicializar proyecto de modelado desde uno ya existente.

Contexto: Se inicializa el proyecto de modelado existente.

Respuesta: Se muestra la interfaz para cargar el proyecto de modelado existente.

3.3 Árbol de utilidad.

Característica	Subcaracterística	Prioridad	Escenario
Eficiencia	Rendimiento	(M,A)	El producto propicia buenos tiempos de respuesta y procesamiento de la información.
Mantenibilidad	Flexibilidad	(M,A)	El producto permite que se le realicen diferentes modificaciones específicas.
	Estabilidad	(M,A)	El producto tiene la capacidad de minimizar los efectos ante cambios que se realicen.
Funcionabilidad	Seguridad	(M,A)	El producto presenta gran capacidad para la protección de la información y los datos.
Confiabilidad	Tolerancia ante fallos	(M,A)	El producto está capacitado para mantener un nivel de ejecución ante cualquier fallo de software.
Usabilidad	Comprensibilidad	(M,B)	El producto presenta características que permiten al usuario entender el software y realizar sus tareas de una manera más entendible.
Portabilidad	Adaptabilidad	(M,M)	El producto presenta capacidades para la adaptación a otros ambientes, tales como Windows y GNU/Linux.
	Instalabilidad	(M,M)	Producto con funciones adecuadas que permiten ser instalada en un ambiente específico.
	Coexistencia	(M,M)	El producto de software posee la capacidad de coexistir con otro software independiente en un ambiente común y compartir los recursos comunes.

Tabla 13: Árbol de utilidad.

3.4 Riesgos identificados y toma de decisiones en función de estos.

Riesgo Identificado	Decisión
Como el ancho de banda es pequeño la conexión es lenta y el tiempo de respuesta y procesamiento de la información es lento.	Este riesgo se puede mitigar al hacer uso de la arquitectura 3 capas, pues esta brinda mantenimiento y soporte sencillo al hacer cambios en las capas, facilitando menores tiempo de respuesta y un mayor procesamiento de la información.
Al realizar cambios en una parte del sistema hay que revisar todo el código para poder realizar los cambios y esto hace que el proceso de cambio es lento.	Este riesgo se puede mitigar al hacer uso de la arquitectura 3 capas, pues como la misma es lineal y permite una separación bien delimitada del código, al realizar algún cambio en una capa, este no afecta a la otra.
La información y los datos del sistema no están bien asegurados.	Este riesgo se puede mitigar al hacer uso de la arquitectura 3 capas, pues esta encapsula los datos, propiciando así una aplicación robusta.

Tabla 14: Riesgos identificados

Conclusiones parciales.

En este capítulo se seleccionaron los atributos de calidad de la norma ISO/IEC 9126 como elementos claves a tener en cuenta para el desarrollo exitoso de una arquitectura sólida. Utilizando el método ATAM y los atributos de calidad antes mencionados para evaluar la arquitectura propuesta, observándose que la arquitectura satisface las necesidades del sistema al detectarse pocos riesgos, dando solución a los mismos mediante decisiones arquitectónicas tomadas.

Conclusiones Generales

El estudio de las herramientas CASE de modelado de software permitieron determinar un conjunto de requisitos con los que debe contar el módulo de modelado.

Se definió la arquitectura del módulo de modelado de software utilizando el patrón arquitectónico 3 capas y los patrones de diseño Experto, Creador, Bajo acoplamiento y Alta cohesión, brindando una manera de proveer la representación de modelos de software utilizando las notaciones UML y BPMN.

Para la evaluación de la arquitectura se utiliza el método ATAM, con los atributos de calidad planteados por la norma ISO-9126, demostrándose la viabilidad de la arquitectura, al detectar pocos riesgos y la mitigación de los mismos.

Recomendaciones

Después del estudio realizado y los resultados obtenidos en el presente trabajo, se recomienda:

- ✓ Utilizar la arquitectura propuesta como base para el desarrollo de la herramienta a personalizar.
- ✓ Emplear el resultado presentado en el trabajo para el aprendizaje de la disciplina de Arquitectura de Software.

Referencia Bibliográfica

1. **Génova, Gonzalo**. Modeling and Metamodeling in Model Driven Development (2009), [en línea], disponible en: www.ie.inf.uc3m.es/ggenova/Warsaw.htm. Consultado: 2012-11-21.

2. **(CEDS) CENTRO DE ESTUDIOS Y DISEÑO DE SISTEMAS,** CURSO BASICO DE ANALISIS Y DISEÑO DE SISTEMAS INFORMATICOS, PRACTICAS CASE CON LA HERRAMIENTA VISIBLE ANALYST – TUTORIALES (2010), [en línea], disponible en: http://ceds.nauta.es/informes/Practicas%20CASE_Ejemplo_Tutorial.pdf Consultado:

3. **Tecuapetla Lara, Francisco Javier.** Capítulo 2. Conceptos de ingeniería de software, CASE y análisis estructurado (2009), [en linea], disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/tecuapetla_l_fj/capitulo2.pdf Consultado: 2012-12-10.

4. **Cruz Matías, Irving Alberto** Herramientas CASE para la generación de código C++ a partir de diagramas de clase UML, (2003), [en línea], disponible en: http://www.utm.mx/~caff/doc/CASEparaGenerarCodigoCconDiagramasUML.pdf Consultado: 2012-12-10.

5. **Instituto Nacional de Estadística e Informática,** Herramientas CASE - Colección Cultura Informática (2009), [en línea], disponible en: www.inei.gob.pe/biblioineipub/bancopub/Inf/Lib5103/Libro.pdf Consultado: 2012-12-10.

6. **Gonzales López, Pascual, Gonzales López, Ana Amelia, Gallud Lázaro, José Antonio**. Herramientas CASE. ¿Cómo incorporarlas con éxito en nuestra organización? (2010), [en línea], disponible en: www.uclm.es/ab/educacion/ensayos/pdf/revista10/10_17.pdf Consultado: 2012-12-10.

7. **Mendez Pozos Pablo Esteban**. Segundo trabajo: Herramienta Case Visual Paradigm | Herramientas Automatizadas, 2012-04-17, [en línea], disponible en: http://dianbeel.blogspot.com/2012/06/segundo-trabajo-herramienta-case-visual.html Consultado 2013-01-24.

8. **González Blanco, Rubén, Pérez Tobalina, Sergio**. LESE-2 Introducción a Rational Rose (2009), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=rational+rose&source=web&cd=9&ved=0CFQQFjAI&url=http%3A%2F%2Fwww.essi.upc.edu%2F~es-

e%2Fweb%2Fdocuments%2Flab%2F0304Q2%2Flessons%2Flese-2%2FLESE-2%2520-%2520Introduccion%2520a%2520Rational%2520Rose.ppt&ei=IvsLUdjjB8Lg0gGXmIDwBg&usg=AFQjCNEz64ZXV5AtmvkH1QHjN5pJT-mhdA&bvm=bv.41867550,d.eWU&cad=rja Consultado 2013-01-24.

9. **Enterprise_Arquitect,** (2012), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=enterprise+architect+documentaci%C3%B3n&source=web&cd=9&ved=0CFoQFjAI&url=http%3A%2F%2Fwww.ecured.cu%2Findex.php%3Ftitle%3DEspecial%3APdfprint%26page%3DEnterprise_Arquitect&ei=yf8LUaDrLIfk9ASEu4H4Bg&usg=AFQjCNFw1nNaMNyNfpMsAdm-3ZZGNrdhSA&cad=rja Consultado 2013-01-25.

10. **Escalona Cuaresma, María José, Gutiérrez Rodríguez, Javier Jesús**. Introducción a Enterprise Architecture (2012), [en línea], disponible en: http://www.lsi.us.es/~javierj/cursos_ficheros/metricaUML/EAintro.pdf Consultado 2013-01-25.

11. **GRACIA OSORIO, JIMMY ALEXANDRO**. ArgoUML (2012), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=argouml%2Bcaracter%C3%ADsticas&source=web&cd=8&ved=0CFoQFjAH&url=http%3A%2F%2Fingenieriasoftwaredos.wikispaces.com%2Ffile%2Fview%2FArgoUML.pptx&ei=7gQMUZG1HY7A9gThwIBg&usg=AFQjCNHARpTOS-kYiJH3pwyYABcJd236Sw&cad=rja. Consultado 2013-01-28.

12. **StarUML.** (2012) [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=staruml&source=web&cd=3&ved=0CD8QFjAC&url=http%3A%2F%2Funiminutotgsandrea.wikispaces.com%2Ffile%2Fview%2FSTARUML.pptx&ei=zAIMUcGHGYP-9QSupoDAAg&usg=AFQjCNHvJpn77fRQWaGAWjKFNrD3cqlScg&cad=rja Consultado 2013-01-28.

13. **ALVAREZ CANO, FRANKLIN, VEGA RAMOS, OMAR.** UMBRELLO (2012), [en línea], disponible en: http://ovruni.files.wordpress.com/2009/11/umbrello-uml-modeller1.pdf Consultado 2013-01-29.

14. **Fouces Lago, Marcos**. Manual de Umbrello UML Modeller (2011), [en línea], disponible en: http://docs.kde.org/stable/es/kdesdk/umbrello/ Consultado 2013-01-29.

15. **Roberts, Harry.** Gaphor, the essence of UML Modelling (2011), [en línea], disponible en: http://gaphor.sourceforge.net/ Consultado 2013-01-30.

16. **Roberts, Harry.** Screenshots (2010), [en línea], disponible en: http://gaphor.sourceforge.net/screenshots.php Consultado 2013-01-30.

17. **Reynoso, Carlos**, Kiccillof Nicolás. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft (2004). [en linea], Disponoble en: www.willydev.net/descargas/prev/Estiloypatron.pdf. Consultado: 2013-02-01
18. **Visconti, Marcello y Astudillo, Hernán,** Fundamentos de Ingeniería de Software (2009), [en línea], dieponible en: inf.utfsm.cl. Consultado: 2013-02-01.
19. **Pérez Mariñán, Martín.** Patrones de Diseño (Design Patterns) (2010), [en línea]. Disponible en: http://www.vico.org/pages/PatronsDisseny/patrones.pdf Consultado: 2013-02-04.
20. **Larman, Caraig.** UML y Patrones. Introducción al análisis y diseño orientado a objeto (2010), [en línea]. Disponible en: http://publidisa.com/PREVIEW-LIBRO-9788483229279.pdf Consultado: 2013-03-12.
21. **CAMACHO, ERIKA, CARDESO, FABIO, NUÑEZ, GABRIEL.** ARQUITECTURAS DE SOFTWARE. (2004) [en línea], disponible en: http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf Consultado 2013-02-01.
22. **García González, Roberto.** Patrones Arquitectónicos de Aplicaciones Empresariales (2008), [en línea], Disponible en: http://ocw.udl.cat/enginyeria-i-arquitectura/enginyeria-del-software-iii/Continguts/2%20-%20Patrones/Patrones%20Arquitectonicos%20de%20Aplicaciones%20Empresariales.pdf Consultado: 2013-03-12.
23. **ccia SCS.** Tema 1. Arquitectura Cliente/Servidor, 3 de octubre de 2008, [en línea], disponible en: http://ccia.ei.uvigo.es/docencia/SCS/Tema1.pdf Consultado: 2013-03-13.
24. **Valdespino Tamayo, Cecilia.** Arquitectura de Plug-in para Sistemas de Visualización Médica (2009), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CC4QFjAA&url=http%3A%2F%2Fdialnet.unirioja.es%2Fdescarga%2Farticulo%2F3919817.pdf&ei=35NQUZz9J-qx0QGK2ICQCg&usg=AFQjCNHcFv82r5Xsvi3qAb2XvstXb4INCw&bvm=bv.44158598,d.dmQ Consultado: 2013-03-13.
25. **Arregui, Miguel.** Tutorial de UML (2009), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=uml+tutorial&source=web&cd=11&ved=0CGwQFjAK&url=http%3A%2F%2Fwww.seis.es%2Fseis%2Finforsalud04%2F2004_Inforsalud_TutorialUML-UP.doc&ei=v4sGUb6lJIvMswb934HYBw&usg=AFQjCNHcMw2fJS1YgYynQDpOmXLsr1hvJQ&bvm=bv.41524429,d.Yms&cad=rja Consultado: 2013- 01-07.

26. **Sparks, Geoffrey.** Una Introducción al UML. El Modelo de Casos de Uso (2010). [en línea], disponible en: http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf Consultado: 2013-03-15.

27. **Popkin Software and Systems,** Modelado de sistemas como UML (2010), [en línea], disponible en: http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf Consultado: 2013-01-08.

28. **Torres Letelier, Patricio.** Desarrollo de software Orientado a Objeto usando UML (2010), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=uml+definicion&source=web&cd=5&ved=0CEYQFjAE&url=https%3A%2F%2Fwww.fillanypdf.com%2FDownload%2FShared%2Fa36b7404-2c07-477f-9da8-fd1995460982%2FEs%2520en%2520PDF.pdf&ei=74cGUfKpCIfJtAaCzYC4CA&usg=AFQjCNHijtXht-Wbml-V2LD-LTUmp6RWyg&bvm=bv.41524429,d.Yms&cad=rja Consultado: 2013-01-09.

29. **ANALITICA**. Sistema de Gestión de Procesos. Manual de diagramas de Procesos bajo estándar BPMN (2010). [en línea], disponible en: http://www.analitica.com.co/website/images/stories/documentosTecnicos_SGP/Manual%20de%20Diagramacion%20de%20Procesos%20Bajo%20Estandar%20BPMN.pdf Consultado: 2013-01-10.

30. **Villarreal, Pablo D.** Modelo Conceptual de Procesos de Negocio BPNM. (2012), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCwQFjAA&url=http%3A%2F%2Fwww.frsf.utn.edu.ar%2Fmatero%2Fvisitante%2Fbajar_apunte.php%3Fid_catedra%3D70%26id_apunte%3D4181&ei=QPeOUYKyJKP54AOukYHQCw&usg=AFQjCNFlPieIO_GqgSAX0SN7gkgpXylS0g&bvm=bv.46340616,d.dmg Consultado: 2013-04-05.

31. **Macías García, Manuel.** GUIA PARA LA IDENTIFICACIÓN Y ANÁLISIS DE PROCESOS. V01 (septiembre de 2007), [en línea], disponible en: http://servicio.uca.es/personal/guia_procesos.pdf Consultado: 2013-04-05.

32. **Wachholtz A, Max y Ciangarotti S, Javier**. Diagrama de Flujo de Datos (2010), [en línea], disponible en: http://alayo.files.wordpress.com/2008/12/diagrama-de-flujo-de-datos2.pdf Consultado 2013-04-05.

33. **González Cagigal, Marco Antonio.** IMPLANTACIÓN DEL RECONOCIMIENTO DE VOZ EN LA UGC DE ENDOCRINOLOGÍA Y NUTRICIÓN (2010), [en línea], disponible en:

http://bibing.us.es/proyectos/abreproy/4245/fichero/PFC+-+Implantaci%C3%B3n+del+Reconocimiento+de+Voz+en+la+UGC+de+Endocr%252F11.Anexo+I.pdf Consultado 2013-04-05.

34. **Garcilaso, Jordana.** Introducción Open UP (2011), [en línea], disponible en: http://www.google.com.cu/url?sa=t&rct=j&q=caracter%C3%ADsticas+openup%2Bmetodolog%C3%ADa&source=web&cd=8&ved=0CFoQFjAH&url=http%3A%2F%2Fbusinesscase.googlecode.com%2Fsvn%2Ftrunk%2Farivadeneira%2FMaterial%2520de%2520Apoyo%2FIntroducci%25C3%25B3n%2520a%2520Open%2520UP.ppt&ei=ExAlUc2gNIHT0wGsw4G4DQ&usg=AFQjCNFU6AiYv5LXN1_DS-IzRojjzTeXhA&bvm=bv.41524429,d.Yms&cad=rja Consultado: 2013-01-12.

35. **Corso, Cynthia Lorena.** Lenguaje Python (2010), [en línea], disponible en: http://labsys.frc.utn.edu.ar/pdf/latinoamerica_educa_III/lenguaje_de_programacion_python.pdf Consultado: 2013-01-18.

36. **Seoane Pascual, Joaquín, González Barahona, Jesús M., Robles, Gregorio**. Introducción al software libre (2011), [en línea], disponible en: http://curso-sobre.berlios.de/introsobre/2.0.1/sobre.html/book1.html Consultado 2013-01-21.

37. **Centro de Estudios y DIseño de Sistemas**. Herramientas CASE- Organizaciones y Fabricantes (2010), [en línea], disponible en: http://ceds.nauta.es/informes/case04.htm Consultado 2013-01-21.

38. **Herramientas CASE**. Lista de Herramientas CASE - Herramientas CASE (2010), [en línea], disponible en: https://sites.google.com/site/herramientascaseudelp/lista-de-herramientas Consultado 2013-01-21.

39. **Qt Designer Manual | Documentation | Qt Project**. QtDesigner, 2012-01-05, [en línea], disponible en: http://qt-project.org/doc/qt-4.8/designer-manual.html Consultado 2013-01-22.

40. Software.com.ar. Visual Paradigm para UML, 2012, [en línea], disponible en: http://www.software.com.ar/visual-paradigm-para-uml.html Consultado 2013-01-22.

41. **Lopez, Patricia.** Instalación y uso básico de BOUML, septiembre 2009, [en línea], disponible en: http://www.ctr.unican.es/asignaturas/MC_OO/Doc/Seminario_BOUML09.pdf Consultado 2013-04-05.

42. **Dávila Nicanor, Leticia.** Evaluación de la Calidad en Sistemas de Información en Internet (2003), [en línea], disponible en: http://www.cs.cinvestav.mx/TesisGraduados/2003/tesisLeticiaDavila.pdf Consulatdo 2013-05-09

43. **Abud Figueroa, María Antonieta**. Calidad en la Industria del Software. La Norma ISO-9126 (2011), [en línea], disponible en: http://www.repositoriodigital.ipn.mx/bitstream/handle/123456789/5321/34-2.pdf?sequence=2 Consultado 2013-05-09.

44. **Delgado Andrea, Castro Alberto, Germán Martín**. Evaluación de Arquitecturas de Software con ATAM (Architecture Tradeoff Analysis Method): un caso de estudio, (2006), [en línea], disponible en: http://www.bvs.hn/cu-2007/ponencias/CAL/CAL027.pdf Consultado 2013-05-09.

Bibliografía consultada

Andrea Delgado, Alberto Castro, Martín Germán. 2006. Evaluación de arquitecturas de software con ATM. 2006.

Arregui, Miguel. Tutorial de UML.

Cagigal, Marco Antonio González. Implantación del reconocimiento de voz en la UGC de endocrinología y nutrición.

Cagugal, Marco Antonio González. Implantación del reconocimiento de voz en la UGC de endocrinología y nutrición.

Carlos Reynoso, Nicolás Kiccillof. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.

Corso, Cynthia Lorena. Lenguaje Python.

Dávila, Leticia. 2003. Evaluación de la calidad en sistemas de información en internet. 2003.

Erika Camacho, Fabio Cardeso, Gabriel Nuñez. 2004. Arquitecturas de software. 2004.

Figueroa, María Antonieta Abud. Calidad en la industria del software. La norma ISO-9126.

Franklin Alvarez Cano, Omar Vega Ramos.

García, Manuel Macías. 2007. Guía para la identificación y análisis de procesos. 2007.

Garcilaso, Jordana. Introducción Open UP.

Génova, Gonzalo. 2009. Modeling and Metamodeling in Model Driven Developement. 2009.

González, Roberto García. Patrones Arquitectónicos Empresariales.

Joaquín Seoane Pascual, Jesus M. González Barahona, Gregirio Robles. Introducción al softwrae libre.

Lago, Marcos Fouces. Manual de Umbrello UML Modeller.

Lara, Francisco Javier Tecuapetla. Conceptos de ingeniería de software, CASE y análisis estructurado.

Larman, Caraig. UML y Patrones. Introducción al análisis y diseño orientado a objeto.

Letelier, Patricio Torres. Desarrollo de software Orientado a Objeto usando UML.

Leteriel, Patricio. Metodologías Ágiles y XP.

Lopez, Patricia. 2009. Instalación y uso básico de BOUML. 2009.

Macías, Manuel García. 2007. s.l. : Guía para la identificación y análisis de procesos, 2007.

Marcello Visconti, Hernán Astudillo. Fundamentos de Ingeniería de Software.

María José Escalona Cuaresma, Javier Jesús Gutiérrez Rodríguez. Introducción a Enterprise Architecture.

Mariñán, Martín Pérez. Patrones de Diseño.

Mary Beth Chrissis, Mike Konrad, Sandy Shrum. Guía para la integración de procesos y la mejora de productos.

Matías, Irving Alverto Cruz. 2003. Herramientas CASE para la generación de código C++ a partir de diagramas de clase UML. 2003.

Max Wachholtz, Javier Ciangarotti. 2008. Diagrama de Flujo de Datos. 2008.

Max Wachholyz, Javier Ciangarotti. Diagrama de Flujo de Datos.

Osorio, Jimmy Alexandro Garcia. ArgoUML.

Pascual López Gonzales, Ana Amelia Gonzales López, José Antonio Gallud Lázaro. Herramientas CASE. ¿Cómo incorporarlas con éxito en nuestra organización?

Patricia Forradellas, Guillermo Pantaleo, Juan Rogers. El modelado CMM-Como garantizar el éxito del proceso de mejoras en las organizaciones, superando los conflictos y tensiones generados por su implementación.

Patricio Letelier, Ma Carmen Panadés. Metodologías ñagiles para el desarrollo de software: XP.

Pozos, Pablo Esteban Mendez. 2012. Herramienta CASE Visual Paradigm. Herramientas Automatizadas. 2012.

Roberts, Harry. Gaphor the essence of UML Modelling.

Rubén Gonzalez Blanco, Sergio Pérez Tobalina. Introducción al Rational Rose.

Sebastián De Hormachea, Francisco Silvera. Análisis de la aplicación de la herramienta Rationl Team Concert para la definición de procesos de software.

Spatks, Geoffrey. Una introducción al UML. El Modelo de Casos de Uso.

Tamayo, Cecilia Valdespino. Arqitectura de Plug-in para sistemas de visualización mádica.

Villareal, Pablo D. 2012. Modelo Conceptual de Proceso de Negocio BPNM. 2012.

Glosario de términos

Árbol de Utilidad: esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

Artefactos: Un artefacto es un producto tangible resultante del proceso de desarrollo de software.

BPMN: Business Process Modeling Notation. Es una notación a través de la cual se expresan los procesos de negocio.

Caso de Uso: Secuencias de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de las secuencias.

CRC: Clase, Responsabilidades, Colaboradores. Es una herramienta principalmente de diseño.

EPC: Cadenas de procesos condicionados por eventos.

Licencia BSD: otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva y permite el uso del código fuente en software no libre.

RTF: es un formato de archivo para el intercambio de documentos multiplataforma.

Stakeholders: Término inglés utilizado por primera vez por R. E. Freeman en su obra: “Strategic Management: A Stakeholder Approach”, (Pitman, 1984) para referirse a quienes pueden afectar o son afectados por las actividades de una empresa.

UML: Lenguaje Unificado de Modelado UML, del inglés, Unified Modeling Language. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

Visual Paradigm: Herramienta de modelado UML y que cuenta con una versión gratuita denominada Community Edition.