Universidad de las Ciencias Informáticas

Facultad 6

Título: "Extensión de la herramienta "*Visual Paradigm for UML*" para la administración de requisitos"

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

Autores: Mayeleiny Pérez Guerra

Raúl Alejandro Rondón Milanés

Tutor: Ing. Alberto Mendoza Garnache

La Habana, junio del 2012

"Año 54 de la Revolución"

“PORQUE PRÁCTICAMENTE TODO, LAS EXPECTATIVAS DE LOS DEMÁS, EL ORGULLO, EL MIEDO AL RIDÍCULO O AL FRACASO SE DESVANECE FRENTE A LA MUERTE, DEJANDO SÓLO LO QUE ES VERDADERAMENTE IMPORTANTE. RECORDAR QUE VAS A MORIR ES LA MEJOR FORMA QUE CONOZCO DE EVITAR LA TRAMPA DE PENSAR QUE TIENES ALGO QUE PERDER. YA ESTÁS DESNUDO. NO HAY RAZÓN PARA NO SEGUIR TU CORAZÓN.”

STEVE JOBS

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmamos la presente a los ___ días del mes de _____ del año 2012.

_____	_____
Mayeleiny Pérez Guerra	Raúl Alejandro Rondón Milanés
Autor	Autor

_____

Ing. Alberto Mendoza Garnache

Tutor

Datos de Contacto

Autores:

Mayeleiny Pérez Guerra
Universidad de las Ciencias Informáticas
La Habana, Cuba
Email: mpguerra@estudiantes.uci.cu

Raúl Alejandro Rondón Milanés
Universidad de las Ciencias Informáticas
La Habana, Cuba
Email: rarondon@estudiantes.uci.cu

Tutor:

Ing. Alberto Mendoza Garnache.
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: agarnache@uci.cu

DEDICATORIA

A mi mamita la persona más querida que tengo a mi lado, que ha sido una madre muy especial y una verdadera amiga. Ha velado por mí aconsejándome y mostrándome el buen camino. Nunca me ha dicho "no". Soy independiente gracias a su educación y determinación de dejarme elegir por mí misma y asumir mis propios riesgos. Te quiero mucho

A mi papito que es el amor de mi vida, quien despertó en mí la creatividad y la lógica, quien me condujo en mis primeros pasos en el mundo. Me ha enseñado de todo un poco, me ha brindado valiosos consejos y he tomado sus opiniones como mías, por confiar tanto en mí y estar seguro que no lo defraudaría. Te quiero mucho.

Mayeleiny Pérez Guerra

A ti Mami, que eres especial y única, cada día de mi carrera tú la has vivido junto a mí, con mucha preocupación, amor incondicional, paciencia y esfuerzo. No existen palabras para describir mi amor por ti y, decir te quiero está demás, en muchas ocasiones has cambiado mi vida por la tuya y hoy te digo que no ha sido en vano. Todo lo que soy te lo debo a ti por ser fuerte y no dejarme sólo en la vida y por darme los consejos que muchas veces me alertaron. A ti Mami.

A mi Papá, por estar pendiente de mí de la mejor manera, por ser una persona con la cual siempre he podido contar, espero que siempre sea así, gracias por confiar en mí, gracias por aconsejarme en todo en la vida, consejos que nunca olvidaré y siempre llevo conmigo.

Raúl Alejandro Rondón Milanés

AGRADECIMIENTOS

En primer lugar quiero brindarle unos especiales agradecimientos a los dos tesoros más importantes en mi vida, mi mamá y mi papá, ya que siempre han estado a mi lado brindándome su apoyo incondicional guiando por el buen camino para que logre todas mis metas.

A mi otra mamita querida Fifi, por el apoyo diario, por extenderme su mano y estar a mi lado en los momentos más difíciles, por escuchar mis lamentos, por estar siempre pendiente de mi, por comprenderme, soportarme, quererme y estar dispuesta a dar un paso al frente cada vez que tengo un problema, eres lo mejor que he conocido en esta universidad ya que he aprendido a quererte día a día por todo el amor que tú me brindas. Muaaaa te quiero mucho.

A Adrian y sus padres que aunque no estemos juntos siempre lo voy a recordar, ya que me supo guiar por el buen camino dándome su apoyo incondicional y por compartir el amor tan lindo que tuvimos.

A mis hermanos en especial a Marcos que siempre ha sido un ejemplo para mí. Gracias por todo el cariño y el amor que me has dado que aunque estés lejos, siempre estás presente en mi corazón. Gracias por darme unos sobrinos tan lindos en especial la niña que es una razón para mí de ser cada día mejor, porque sé que donde quiera que se encuentre se refleja en su tía tan querida.

A mis tías(os), primas(os) por su apoyo y cariño en todo momento. En especial a mi tía Tere por quererme como una hija más y mi tío Eugenio que aunque no estés aquí sé que me cuidas y me guías.

A mis Abuelos en especial a mi abuelita querida que siempre está pendiente de mí y de mi madre.

A mi tutor por brindarnos todos sus conocimientos y apoyo para la realización de este trabajo que aunque nos peleaba tanto siempre lo hizo por nuestro bien.

A mis amigos, por los momentos que pasamos juntos y los recuerdos que quedaran grabados, en especial a él clan o las chicas súper poderosas Mayelín y Ayeris por vivir tantas experiencias juntas en todos estos años que nunca olvidaré. A Ivettte y Yailen mis amigas del alma por escucharme, aconsejarme y guardar todos mis secretos, ayudarme, por ser pacientes y comprenderme.

A Reinier mi profe querido ya que me ha sabido aconsejar y me ha ayudado en todo lo que he necesitado sin ningún pretexto al igual que Baby que siempre me ha brindado una sonrisa incondicional.

Y por último quisiera agradecerle a mi compañero de tesis por escuchar mis lamentos, por estar siempre pendiente de mí, por comprenderme, soportarme y quererme, muchas gracias por todos los momentos que pasamos juntos en la realización de esta tesis en la cual fuiste un guía y pudiste trasmitirme muchos de tus conocimientos, espero de todo corazón que triunfes en la vida ya que tú te lo mereces.

Mayeleiny Pérez Guerra

A mis padres por estar siempre junto a mí en los momentos buenos y malos durante esta carrera, por la esperanza y la confianza depositada, por el amor y el apoyo. Todo lo que hoy soy se lo debo a ustedes, lo hicieron muy bien. Gracias.

A Manuel Velásquez por ser un amigo siempre en todo momento, para lo que fuera necesario. Gracias.

A Elianet por tu apoyo incondicional y tanto amor. Muchas Gracias.

A todos los compañeros(as) que he conocido durante estos 5 años, compañeros de aula, de apartamento y otros conocidos también, todos han sido importantes porque me enseñaron algo cada uno y no es posible olvidar a personas tan especiales como ustedes. A mis amigas Mayelin y Ayeris. Gracias.

A los profesores que hicieron posible que hoy esté aquí, por el conocimiento brindado. Gracias.

A mi tutor, por la paciencia y conocimiento brindado. Gracias.

A mi compañera de tesis, por estar siempre conmigo, por los momentos que pasamos durante todo el período, en el cual aprendimos juntos muchas cosas que estoy seguro que no se olvidarán. Y aquí

estamos… "No sé si es importante, pero nunca es demasiado tarde para ser quien quieres ser, no hay límites de tiempo, puedes empezar cuando quieras, puedes cambiar o seguir siendo la misma, no hay reglas para tal cosa, puedes hacer o echar a perder todo. Espero que hagas lo mejor, espero que veas cosas que te asombren, espero que sientas cosas que nunca sentiste antes, espero que conozcas gente con un punto de vista diferente, espero que vivas una vida de la que te sientas orgullosa y si encuentras que no… espero que... tengas fuerzas para empezar de nuevo." … Suerte…

Raúl Alejandro Rondón Milanés

RESUMEN

En la Universidad de las Ciencias Informáticas (UCI) se desarrollan un número significativo de proyectos en centros de desarrollo de software vinculados a las facultades. El Centro de Tecnología de Gestión de Datos (DATEC) es uno de ellos, el mismo desarrolla sus productos utilizando la herramienta CASE *"Visual Paradigm for UML"* como apoyo para la realización del análisis del desarrollo de software. Esta investigación surge producto al inconveniente que posee la herramienta *"Visual Paradigm for UML"* de no proveer la funcionalidad de Administración de Requisitos. El presente trabajo tiene como objetivo desarrollar una extensión de la herramienta *"Visual Paradigm for UML"* para la administración de requisitos. Con esta extensión se pretende reducir el tiempo para realizar el análisis de los requisitos, garantizar un alto nivel en la calidad de la aplicación de acuerdo con los intereses del cliente, así como aprovechar las herramientas informáticas definidas para el desarrollo de software en el Centro. Como resultado se obtuvo un prototipo funcional de extensión de la herramienta CASE *"Visual Paradigm for UML"*. La extensión implementada se sometió a pruebas funcionales mediante las cuales se comprobó el correcto funcionamiento de la misma. El resultado alcanzado es de suma importancia pues beneficia directamente el proceso de desarrollo de software en DATEC.

PALABRAS CLAVES:

"Visual Paradigm for UML", extensión, DATEC

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Durante los primeros años de la era de las computadoras estas eran en sí un conglomerado de componentes electrónicos, conocidos hoy como hardware, sin embargo, para que estos componentes funcionaran de forma ordenada y a la vez poder aprovechar todos los recursos que esta brindaba de manera que se pudiera explotar al máximo su capacidad, se creó el software, el cual está formado por una serie de instrucciones lógicas y datos, que permiten la interacción usuario y máquina.

En sus inicios los sistemas informáticos eran pequeños y satisfacían solo objetivos específicos; la mayoría se desarrollaban y eran utilizados por la misma persona u organización. La misma persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba. El diseño era un proceso implícito, realizado en la mente de alguien y, la documentación normalmente no existía. El desarrollo del software se realizaba virtualmente sin ninguna planificación, hasta que los planes comenzaron a romperse y los costes a correr. Los programadores trataban de hacer las cosas bien, y con esfuerzo, a menudo salían con éxito. El software se diseñaba a medida para cada aplicación y tenía una distribución relativamente pequeña, aún así ganaban en tiempo y recursos. Luego empezaron a adentrarse en la mayoría de negocios y procesos comerciales de la vida social del hombre.

Esta crecida propició un auge en la variedad de los software, los mismos eran cada vez más complejos debido a que los flujos de trabajos eran muy variables en la mayoría de los centros laborables y estos querían informatizar más procesos de su gama de servicios. Para dar soporte a estos software e incrementar las nuevas prestaciones surgieron nuevas disciplinas, entre ellas se encuentra: la Ingeniería de Software, esta garantiza en gran medida el éxito de los proyectos informáticos ya que brinda la posibilidad de definir los requisitos de un sistema. Estos últimos constituyen los cimientos principales de cualquier producto de software, por lo que se han convertido en centro de atención para muchos ingenieros.

Cuba no ha estado al margen de esta crecida mundial y ha volcado gran parte de su desarrollo y recursos a la informatización de algunos de sus renglones económicos. La Universidad de las Ciencias Informática (UCI) como pilar fundamental de este proceso ha concentrado su fuerza de trabajo en la informatización primeramente de los sectores principales de la economía del país. La UCI ha elevado la calidad del desarrollo de software, por lo que la misma ha creado una infraestructura para hacer posible este proceso dentro de las cuales se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC), el cual tiene la misión de crear bienes y servicios informáticos relacionados con la gestión de datos, área del conocimiento que agrupa tanto a los sistemas de información, como a los denominados sistemas de inteligencia empresarial o de negocios, cuyo propósito fundamental es apoyar el proceso

de toma de decisiones (1), además destacar que el Centro es especializado en tecnologías de Gestión de Datos y Bioinformática con prestigio nacional e internacional, reconocido por sus soluciones integrales, productos y servicios, con una activa colaboración con empresas, centros de investigación y desarrollo de alto nivel, así como con comunidades internacionales afines. Para una mejor organización, el Centro está dividido en diferentes departamentos, una de ellas es el departamento de Integración de Soluciones el cual se dedica a la creación de productos y servicios que incluyen: productos para la gestión de proyectos y específicamente la gestión de proyectos de software. Servicios de integración de soluciones. Desarrollo de las herramientas necesarias para crear soluciones integrales de Análisis de Datos.

Dicho departamento tiene definida una estructura organizativa divida en tres grupos fundamentales: Análisis, Arquitectura y Desarrollo. El Grupo de Análisis es el encargado de realizar actividades en el tiempo establecido y con la calidad requerida. Los analistas son el puente de comunicación entre los clientes y los desarrolladores, además se encargan de validar que se satisfagan las necesidades de los usuarios finales durante todo el desarrollo del proyecto. El departamento de Integración de Soluciones se dedica a la creación de productos y servicios que conforman en su conjunto la Plataforma de Apoyo a la Toma de Decisiones y Soluciones Integrales (PATDSI) la cual incluye: productos para la gestión de proyectos de software, servicios de integración de soluciones y desarrollo de las herramientas necesarias para crear soluciones integrales de análisis de datos, el departamento también incluye en su desarrollo tecnológico productos como libros web, plugin realizados para mejorar el trabajo. La propuesta final de esta investigación tiene como alcance además de los definidos extenderse por los demás departamentos del Centro.

PATDSI dispone de varios productos tales como: el Generador Dinámico de Reportes (GDR), el Sistema Integral de Gestión Estadística (SIGE), el Sistema de Información de Gobierno (SIGOB), el Sistema Integral de Gestión de Datos (SIGDAT) entre otros.

En materia de desarrollo de software, la Dirección de Calidad de Software de la UCI establece políticas organizacionales (2) que guían el proceso de desarrollo de software con el objetivo de producir software con la calidad requerida, cumpliendo con el nivel dos de CMMI, el cual define siete áreas de procesos tales como: aseguramiento de la Calidad a Procesos y Productos (PPQA), Administración de la Configuración (CM), Planeación del Proyecto (PP), Monitoreo y Control del Proyecto (PMC), Medición y Análisis (MA), Administración de Acuerdos con Proveedores (SAM) y Administración de Requisitos (REQM).

La Administración de Requisitos tiene como propósito administrar los requisitos del proyecto e identificar inconsistencias entre los mismos, los subproductos y los planes del proyecto. Define como política que: en los proyectos de la organización, los requisitos a desarrollar y los cambios introducidos a estos durante el ciclo de vida sean entendidos, identificados, controlados y traceados para contribuir al correcto desarrollo del producto final. Se debe mantener la trazabilidad de los requisitos con sus requisitos derivados y los productos de trabajo de forma bidireccional, asegurando la realización de todos los requisitos y la documentación de los mismos. Se debe contar con una especificación clara y completa desde las fases iniciales para no tener problemas posteriores que implicarían un retraso en el cronograma, un presupuesto erróneo, o hasta la posible cancelación del proyecto. Es importante que el documento que se obtenga de esta etapa sea un reflejo real del acuerdo de las partes involucradas.

El proceso de administración de requisito en el Centro no está automatizado ya que no existe una herramienta capaz de realizar este proceso para mejorar el trabajo de los analistas que llevan a cabo dicho proceso por medio de un fichero de formato Excel el cual se torna muy engorroso. La Dirección de Calidad de Software de la UCI propone utilizar dos herramientas para realizar esta actividad, dichas herramientas son:

OSRMT y Enterprise Architect, las cuales no se adaptan a las necesidades del Centro ya que la primera presenta problemas de instalación, no es muy interactiva para que el usuario trabaje de la mejor manera, algunas funcionalidades no se han terminado y otras están a medias, de la misma manera es válido resaltar que no es posible generar automáticamente un documento de requisitos para entregar al cliente, la segunda herramienta propuesta es lo suficientemente capaz de llevar a cabo este proceso pero posee una desventaja que la lleva a ser descartada por el grupo de trabajo, esta desventaja es que es un software privativo.

La propuesta presentada en este documento es capaz de lidiar con todas estas desventajas que presentan las recomendadas por la Dirección de Calidad de Software de la UCI, y se integra con la herramienta CASE utilizada en el Centro, de manera que es lo suficientemente portable para ser utilizada dentro y fuera del departamento, cumpliendo con especificaciones del proceso de administración de requisito.

En aras de cumplir con las políticas antes mencionadas, debido a la complejidad funcional, a la importancia de los procesos de negocio que se automatizan, a las dimensiones considerables de los productos resulta engorroso llevar a cabo todo el proceso de administración de los requisitos de los productos de software que se desarrollan en el departamento de Soluciones Integrales, por lo que se

plantea como **problema científico:** ¿Cómo contribuir con el proceso de administración de requisitos de los productos de software?

La investigación tiene como **objeto de estudio:** la ingeniería de requisitos enmarcado en el **campo de acción:** el proceso de administración de requisitos. Para darle solución al problema planteado se define como **objetivo general:** desarrollar una extensión de la herramienta *"Visual Paradigm for UML"* para la administración de requisitos.

En correspondencia con ello, se plantean como **objetivos específicos:**

- ✓ Identificar las funcionalidades de la extensión de la herramienta CASE *"Visual Paradigm for UML".*
- ✓ Realizar el diseño de la extensión a partir de los requisitos identificados.
- ✓ Implementar las funcionalidades definidas.
- ✓ Realizar pruebas que demuestren el correcto funcionamiento de la extensión.

Para lograr el correcto cumplimiento del objetivo se proponen las siguientes **tareas investigativas:**

- ✓ Revisión bibliográfica de las herramientas existentes y de los enfoques teóricos de la administración de requisitos.
- ✓ Revisión bibliográfica de la documentación técnica de *"Visual Paradigm for UML"* referida a la extensión de la herramienta.
- ✓ Realización del modelado del dominio.
- ✓ Definición de los requisitos funcionales y no funcionales para el correcto funcionamiento de la extensión de *"Visual Paradigm for UML".*
- ✓ Confección del modelo de casos de uso del sistema de la extensión.
- ✓ Elaboración del modelo de diseño.
- ✓ Elaboración del modelo de implementación.
- ✓ Diseño de los casos de uso definidos a partir de los requisitos funcionales.
- ✓ Realización de pruebas funcionales para comprobar el correcto funcionamiento de la extensión.

El presente documento se divide en varios capítulos:

Capítulo 1. Estudio Teórico y Tecnológico del desarrollo de la extensión: en este capítulo se realiza un estudio relacionado a la ingeniería de requisitos, sus principales procesos, se estudian las principales herramientas existentes para la administración de requisitos. Se analizan los elementos arquitectónicos a tener en cuenta para la implementación de extensiones para *"Visual Paradigm for UML".* Además se definen las principales herramientas y tecnologías necesarias para desarrollar la extensión de la herramienta.

Capítulo 2. Análisis y diseño de la extensión: en este capítulo se define todo lo referente a las funcionalidades que debe garantizar la extensión de la herramienta. Se construyen los artefactos asociados al proceso de análisis y diseño, como parte de análisis y modelando la estructura y comportamiento de la extensión.

Capítulo 3. Implementación y prueba de la extensión: en este capítulo se presenta los artefactos asociados a las disciplinas de implementación y prueba. Se elabora el modelo de implementación a través de los diagramas de componentes. Además se realizan las pruebas para garantizar el correcto funcionamiento de la extensión implementada.

Fundamento Teórico y Tecnológico para el desarrollo de la extensión

En este capítulo se abordará todo lo referente al estudio de las tendencias, técnicas y tecnologías usadas en la actualidad y que servirán de apoyo para la solución del problema al que se enfrenta. Todo con el objetivo fundamental de brindar información complementaria para propiciar un mejor entendimiento de los capítulos venideros.

1.1 Ingeniería de Requisitos del Software

¿Qué es un requisito? Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal. Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste.

Los requisitos en sí, constituyen el enlace entre las necesidades reales de los clientes, usuarios y otros participantes vinculados al sistema; consisten en un conjunto de actividades y transformaciones que pretenden comprender lo que se desea para la realización de un software, y deben quedar oficialmente documentados.

El propósito de la Ingeniería de Requisitos es hacer que los requisitos alcancen un estado óptimo antes de alcanzar la fase de diseño en el proyecto. Existen gran variedad de conceptos que definen la ingeniería de requisitos expresando que es un proceso durante el cual se hace una compilación, análisis y verificación de las necesidades del cliente para un sistema. Cuya meta es entregar una especificación de requisitos de software correcta y completa.

“La Ingeniería de Requisitos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software”. (3)

La ingeniería de requisitos es el proceso de desarrollar una especificación de software. Las especificaciones pretenden comunicar las necesidades del cliente a los desarrolladores del sistema.

1.2 Procesos de la Ingeniería de Requisitos

Estudio de Viabilidad: la entrada de este es un conjunto de requisitos de negocios preliminar, una descripción resumida del sistema y de cómo éste pretende contribuir a los procesos de negocio. Los

resultados del proceso de viabilidad debería ser un informe que recomiende si merece o no la pena seguir con la ingeniería de requisitos y el proceso de desarrollo del sistema.

Obtención y Análisis: en esta actividad los ingenieros del software trabajan con los clientes y los usuarios finales del sistema para determinar el dominio de la aplicación, que servicios debe proporcionar el sistema, el rendimiento requerido del sistema y la restricción del hardware.

Especificación: es el proceso de recoger información sobre el sistema propuesto y los existentes y extraer los requerimientos del usuario y de la información.

Validación: coincide parcialmente con el análisis ya que este implica encontrar problemas con los requisitos. La validación de los mismos es importante debido a que los errores en el documento pueden conducir a importantes costes al repetir el trabajo cuando son descubiertos durante el desarrollo o después de que el sistema este en uso.

Administración: los requisitos para sistemas de software grandes son siempre cambiantes. Una razón es que estos sistemas normalmente se desarrollan para abordar problemas. Debido a que el problema no puede definirse completamente, es muy probable que los requisitos del software sean incompletos. Estos requisitos deben entonces evolucionar para reflejar esta perspectiva cambiante del problema.

Además, una vez que un sistema se ha instalado, inevitablemente surgen nuevos requisitos. Es difícil para los usuarios y clientes del sistema anticipar qué efectos tendrá el sistema nuevo en la organización. Cuando los usuarios finales tienen experiencia con un sistema, descubren nuevas necesidades y prioridades:

- ✓ Normalmente, los sistemas grandes tienen una comunidad de usuarios diversa donde tienen diferentes requisitos y prioridades. Estos pueden contradecirse o estar en conflicto. Los requisitos finales del sistema son inevitablemente un compromiso entre ellos y con la experiencia, a menudo se descubre que la ayuda suministrada a los diferentes usuarios tienen que cambiarse.
- ✓ Las personas que pagan por el sistema y los usuarios de este raramente son la misma persona. Los clientes del sistema imponen requisitos debido a las restricciones organizacionales y de presupuesto. Estos pueden estar en conflicto con los requisitos de los usuarios finales y, después de la entrega, pueden tener que añadirse nuevas características de apoyo al usuario si el sistema tiene que cumplir sus objetivos.
- ✓ El entorno del negocio y técnico del sistema cambia después de la instalación y estos cambios se deben reflejar en el sistema. Se puede introducir nuevos hardware, puede ser necesario que

el sistema interactúe con otros sistemas, las prioridades de negocio pueden cambiar con modificaciones consecuentes en la ayuda al sistema, y puede haber una nueva legislación y regulaciones que deben ser implantadas por el sistema.

La gestión es el proceso de comprender y controlar los cambios del sistema. Es necesario mantenerse al tanto de los requisitos y mantener vínculos entre los dependientes de forma que se pueda evaluar el impacto de los cambios. Hay que establecer un proceso formal para implementar las propuestas de cambio y vincular estos al sistema. (4)

Con el objetivo de controlar la complejidad, riesgo, alcance del proyecto, y definir los roles y criterios para un software o un proyecto de negocio exitoso, las herramientas para la administración de requisitos pueden proveer además una trazabilidad completa hasta los entregables finales y el comportamiento del sistema, el uso de estas herramientas tienen como finalidad mejorar la productividad y calidad en el desarrollo de los proyectos de software.

1.3 Herramientas para la Administración de Requisitos

Actualmente en el mundo de la producción de software la Ingeniería de Requisitos está muy presente al cumplir un papel importante en el proceso de desarrollo de los mismos. La Administración de Requisitos constituye uno de los procesos que implica a su vez el desarrollo de varias actividades de vital importancia que ofrece este proceso y últimamente en el mercado de software se ha dado un gran avance en la implementación de herramientas de administración de requisitos. Dentro de estas se encuentran:

- ✓ RequisitePro
- ✓ Gatherspace
- ✓ OSMRT
- ✓ Enterprise Architect

RequisitePro: es una solución fácil de usar, es una herramienta de administración de requisitos que le permite al equipo crear y compartir sus requisitos utilizando métodos basados en documentos potenciados por la aplicación de las capacidades de una base de datos, tales como la trazabilidad y análisis de impacto. El resultado es una mejor comunicación y administración de requisitos con una mayor probabilidad de completar los proyectos en tiempo, dentro del presupuesto y superando las

expectativas. Los proyectos exitosos comienzan con una buena administración de requisitos cuanto más efectiva sea su ejecución, mayor será el resultado en calidad y satisfacción del cliente.

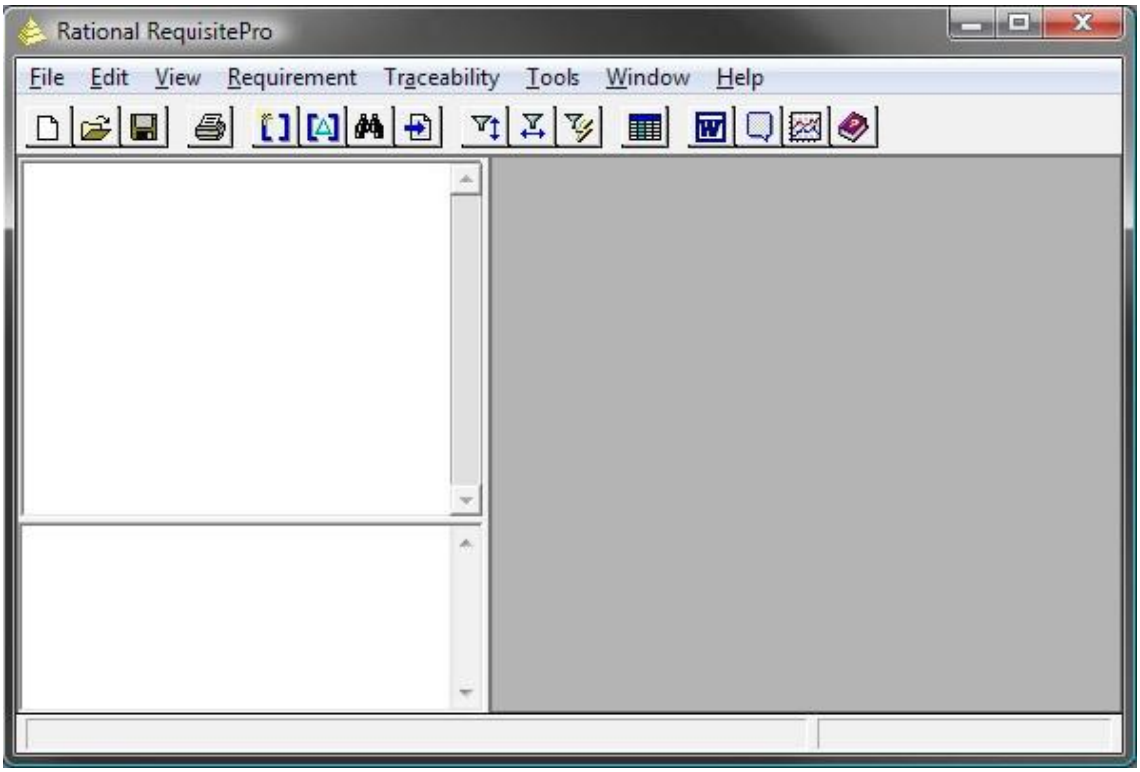


Ilustración 1 Herramienta de Administración de Requisito RequisitePro

- **Integración de Word y Base de Datos**

Soporta Microsoft Word para creación y comunicación de los requisitos; complementa las entradas en forma de documentos con una base de datos comercial para agregar capacidades de organización, seguimiento y administración.

- **Potente motor de requisitos**

Se pueden configurar fácilmente los tipos de requisitos, atributos y tipo de documentos. Definir consultas y filtros para encontrar rápidamente la información de interés.

- **Trazabilidad**

Se pueden configurar y dar seguimiento a las relaciones entre requisitos para verificar que los requisitos de alto nivel están representados dentro de las especificaciones detalladas de requisitos de software.

- **Acceso vía Web mediante RequisiteWeb**

Todo aquel que posea acceso Web, independientemente de la plataforma, puede ver y modificar rápida y eficientemente los requisitos sin necesidad de tener cargado el RationalRequisitePro en su máquina.

Ventajas

- ✓ Propicia una mejor comunicación, mejoras en el trabajo en equipo y reduce el riesgo de los proyectos.
- ✓ Proporciona a los equipos la posibilidad de comprender el impacto de los cambios.
- ✓ Garantiza que todos los componentes del equipo estarán informados de los requisitos más actuales para asegurar la coherencia.
- ✓ Proporciona acceso basado en web para los equipos distribuidos.

Desventajas

- ✓ El programa es sólo para Windows y es privativo.
- ✓ No ofrece soporte a pruebas, es necesario utilizar herramientas externas.

Gatherspace: es una herramienta de entorno web para la administración de requisitos de software y de gran utilidad para trabajo de equipos distribuidos.

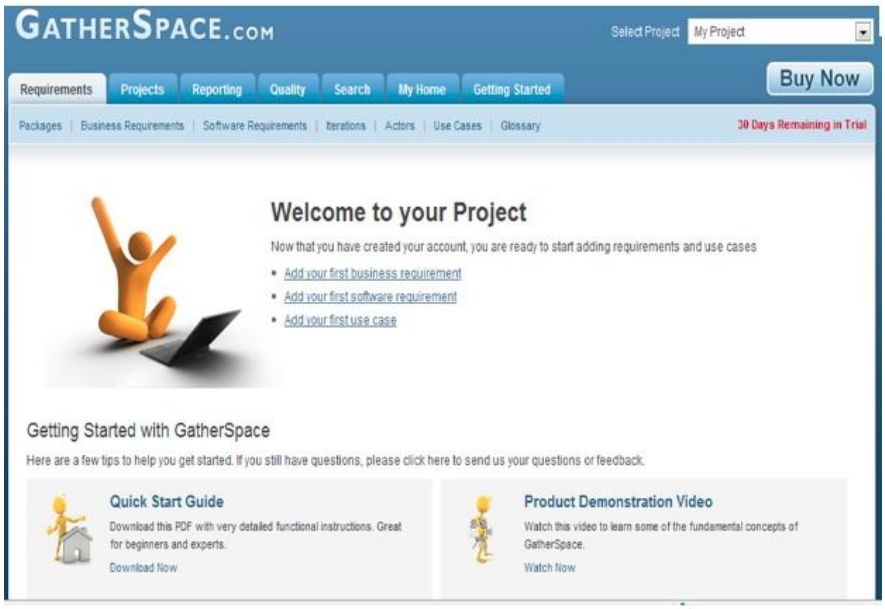


Ilustración 2 Herramienta de Administración de Requisito Gatherspace

Ventajas

- ✓ La herramienta cuenta con un módulo de requisitos donde se gestionan los paquetes, características, requisitos de software, iteraciones, actores, casos de uso y glosarios.
- ✓ Módulo para gestionar proyectos y para explotar la información contenida en el proyecto.
- ✓ Módulo de generación de reporte (Visión y Requisitos, Modelo de Casos de Uso, Detalles de Características y Requisitos Asociados, Casos de Uso, Detalles de Requisitos, Trazabilidad, Cambios y Errores).

Desventajas

- ✓ La aplicación no es un paquete de instalación por lo tanto corre en servidores fuera del lugar de trabajo.
- ✓ Es una herramienta privativa.

OSMRT 4.0: se trata de una herramienta de gestión de requisitos, que permite la descripción avanzada de diversos tipos de requisitos y garantiza la trazabilidad entre todos los documentos relacionados con la ingeniería de requisitos (funcionalidades, requisitos, casos de uso, casos de prueba).

Ventajas

- ✓ La visualización de requisitos en forma jerárquica es intuitiva y fácil de manejar.
- ✓ Existen diversas distribuciones, tanto para un equipo en local como para un servidor de aplicaciones para permitir desarrollo colaborativo.
- ✓ Su licencia es GPL.
- ✓ Es un desarrollo basado en Java, por lo que es multiplataforma.
- ✓ Las nuevas versiones incorporan un cliente Web para permitir accesos desde internet.
- ✓ Como herramienta Open Source de gestión de requisitos no tiene mucha competencia en cuanto a la funcionalidad ofrecida.
- ✓ Tiene una buena documentación pese a tratarse de una herramienta muy reciente.
- ✓ A pesar de ser llevada a cabo por un único desarrollador, el ritmo de mejoras y nuevas versiones es constante a lo largo del último año.
- ✓ Existen muchas opciones para configurar y personalizar la herramienta a las necesidades concretas de una organización.
- ✓ Lleva incorporado un sistema de gestión de la configuración que permite definir líneas base.

- ✓ Existe un gran soporte para mantener la trazabilidad entre los documentos.
- ✓ Existen mecanismos que facilitan la importación y exportación de la información en XML.

Desventajas

- ✓ Es un desarrollo llevado a cabo por una persona individual, por lo que existe el riesgo de que no sea sostenible a lo largo del tiempo.
- ✓ No existe un soporte empresarial.
- ✓ Las nuevas versiones no están planificadas ni se anuncian claramente las mejoras que serán incorporadas. Es posible que las nuevas versiones no sean compatibles con las anteriores.
- ✓ No es posible generar automáticamente un documento de requisitos para entregar al cliente.
- ✓ Algunas funcionalidades no han sido desarrolladas completamente y están a medias.
- ✓ La interfaz de usuario es en ocasiones lenta.
- ✓ Se ofrecen pocos mensajes de confirmación y aviso al usuario (la interacción con el usuario es pobre).

Enterprise Architect (EA) 7.5: es una herramienta CASE (Computer Aided Software Engineering) para el diseño y construcción de sistemas de software. EA soporta la especificación de UML 2.0 que describe un lenguaje visual por el cual se pueden definir mapas o modelos de un proyecto. EA es una herramienta progresiva que cubre todos los aspectos del ciclo de desarrollo y proporciona una trazabilidad completa desde la fase inicial del diseño a través del despliegue y mantenimiento. También provee soporte para pruebas, mantenimiento y control de cambio.

Ediciones Standard		Precio Base
Corporativa	Soporta grandes equipos colaborativos con acceso remoto a DBMS y seguridad.	US $239
Profesional	Modelado UML completo para grupos de trabajo, analistas y desarrolladores.	US $199
Escritorio	Herramienta de modelado UML exhaustivo para analistas individuales	US $135

Ediciones Suite		Precio Base
Ultimate	La experiencia Enterprise Architect completa a través de múltiples dominios!	US $699
Ingeniería de sistemas	Para ingeniería de sistemas, sistemas embebidos y en tiempo real.	US $599
Ingeniería de software y negocio	Para modelado de negocio, arquitectura, desarrollo y más.	US $599

Ilustración 3 Herramienta de Administración de Requisito Enterprise Architect

Algunas de las características claves de Enterprise Architect son:

- ✓ Crea elementos del modelo UML para un amplio alcance de objetivos.
- ✓ Ubica esos elementos en diagramas y paquetes.
- ✓ Documenta los elementos que se han creado. La documentación de alta calidad puede ser rápidamente exportada desde sus modelos en formato RTF e importar a Word para una personalización y presentación final.
- ✓ Se puede realizar ingeniería reversa del código existente en varios lenguajes tales como: C++, C#, Delphi, Java, Python, PHP, VB.NET y clases de Visual Basic.
- ✓ Permite estimar el tamaño de un proyecto en esfuerzo de trabajo en horas. (5)

A continuación se presenta una tabla comparativa entre las herramientas estudiadas, en la misma se puede evidenciar las características que presenta cada una.

Herramientas/Características	Privativa	Interfaz	Portable	Multiplataforma
Enterprise Architect	Si	Amigable	No	Si
Gatherspace	Si	Amigable	No	Si
RequisitePro	Si	Amigable	No	No
OSRMT	No	Poco Amigable	Si	Si
Extensión para Administración de Requisitos	No	Amigable	Si	Si

Tabla 1 Tabla comparativa entre herramientas

Después de analizar las herramientas y de conocer las funcionalidades que cada una ofrece, se pueden identificar los requisitos funcionales y no funcionales que tendrá la herramienta que guía la presente investigación. Por todo lo antes expuesto para la realización de la herramienta de este trabajo, será un producto libre y de código abierto que cumpla con las especificaciones del proceso de administración de requisitos, además de integrarse con la herramienta CASE utilizada en el Centro, ser portable y reutilizable por otros departamentos y Centros.

1.4 Plataforma Tecnológica

Metodología de desarrollo de software

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto de software. Se definen una serie de pasos a seguir para obtener un software de calidad. Las

metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas abarcan procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software. Sus procesos se descomponen a nivel de tareas o actividades elementales, donde cada tarea está identificada por un procedimiento que define la forma de llevarla a cabo. (6)

OpenUP: Carmina Lizeth Torres Flores y Germán Harvey Alférez Salinas en un escrito acentúan que OpenUP (Open Unified Process) es un proceso de desarrollo iterativo del software que es mínimo, completo y extensible. El proceso es mínimo porque solamente el contenido fundamental es incluido; es completo puesto que puede ser manifestado como todo el proceso para construir un sistema; extensible a causa de que puede ser utilizado como fundamento sobre el cual el contenido de proceso se pueda agregar o adaptar según lo necesitado. Como metodología de desarrollo es conducida por el principio de colaboración para alinear intereses y para compartir su comprensión. (7)

Es una versión más ágil de lo que es RUP (Rational Unified Process). OpenUP plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo corto. Principalmente plantea que se deben utilizar solo los procesos que sean necesarios y en este caso se necesita de personas y profesionales que sean capaces. Además esboza que no se deben utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario y puede ser modificado, mejorado y extendido.

Después de analizar esta metodología de desarrollo de software se decidió trabajar con la misma, debido a que está definida por la arquitectura del departamento, o sea es una decisión arquitectónica tomada por los desarrolladores del departamento, la que más se adapta a las necesidades del equipo de desarrollo y en la que más experiencia se tiene. Además constituye un marco de trabajo genérico que puede ser adaptado a una gran diversidad de sistemas de software independientemente del tamaño del proyecto, el tipo de organización y los diferentes niveles de aptitud. Es una metodología adaptable al contexto y a las necesidades que permite tener bien claro y accesible el proceso de desarrollo que se quiere alcanzar.

Lenguaje de programación

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se pone a disposición del programador para que éste pueda comunicarse con los dispositivos de hardware y software existentes.

Java: es un lenguaje de programación sencillo, orientado a objetos, de propósito general e independiente de la plataforma de desarrollo. Fue creado por James Gosling, cumpliendo con los términos de la compañía Sun Microsystems "compilar una vez y ejecutar en cualquier parte". (8)

Java es muy amplio y variado. Posibilita la “integración externa”, como el uso de herramientas, métodos y funcionalidades desarrolladas por otros programadores, que supone una ventaja tanto en el ámbito desarrollo como en la repercusión final de un proyecto.

Las características principales de Java respecto a otros lenguajes de programación son:

- ✓ Simple: ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje.
- ✓ Orientado a objetos: java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.
- ✓ Robusto: realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria.
- ✓ Seguro: las aplicaciones de Java no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción con virus.
- ✓ Portable: como el código compilado de Java es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java.
- ✓ Arquitectura neutral: el compilador Java compila su código a un fichero objeto de formato independiente a la arquitectura de la máquina en que se ejecutará. De esa manera Java logra ser un lenguaje que no depende de una arquitectura computacional definida.
- ✓ Dinámico: no requiere que se compilen todas las clases de un programa para que este funcione.

Es muy importante que los sistemas de gestión de información puedan generar reportes a partir de los datos que perduran en la base de datos y en la aplicación, ya que de esta manera los usuarios pueden hacer un análisis más sofisticado y práctico de la información incluso en el momento de tomar decisiones. La generación de reportes sin dudas ayuda a disponer de la información de una manera más rápida y cuenta con medios más flexibles para distribuirla, el lenguaje utilizado tiene la posibilidad de trabajar con distintos tipos ficheros los cuales son aptos para realizar reportes al usuario.

¿Qué es un reporte? Un reporte o informe es una manera organizada de mostrar información procedente de una fuente de datos.

El lenguaje de programación Java posee bibliotecas para el trabajo con fichero de extensión PDF y XLS, a través de estos se puede editar y modificar las propiedades y las características de dichos archivos y además te permite automatizar todos los procesos desde tus propias aplicaciones. Con estas bibliotecas podrás elegir entre distintas opciones, generar documentos dinámicos desde ficheros XML o desde bases de datos, añadir bookmarks, insertar numeración a las páginas y nombres, aplicar marcas de agua, etcétera. También se logra editar el PDF cortando, concatenando y manipulando diferentes páginas del formato entre sí.

Teniendo en cuenta las características antes mencionadas de Java, y además que *"Visual Paradigm for UML"* es apoyado por un conjunto de idiomas tanto en la generación del código como en la Ingeniería Inversa por mencionar un ejemplo, tiene la capacidad de soportar el lenguaje Java especialmente por ser multiplataforma sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. También es el lenguaje en el que está desarrollada la herramienta CASE por lo que sus extensiones deben seguir el mismo patrón de desarrollo. Además de que en la actualidad es muy utilizado ya que permite el desarrollo de programas seguros y de alto rendimiento, el colectivo de autores decide utilizarlo como lenguaje de programación.

Herramientas a utilizar

Un Entorno Integrado de Desarrollo en inglés Integrated Development Enviroment (IDE) es un programa compuesto por un conjunto de herramientas para un desarrollador que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Dentro de los más populares a nivel mundial se encuentran el Eclipse y el NetBeans. (9)

NetBeans: Herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrita en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. Es un producto libre y gratuito sin restricciones de uso. (10)

Soporta varios lenguajes, entre ellos:

- ✓ Java
- ✓ C/C++
- ✓ Ruby on Rails
- ✓ PHP

Características de NetBeans:

- ✓ Instalación y actualización simple.

- ✓ Diseñador visual de formularios para Swing GUI.
- ✓ Profiling integrado, profiling “points”.
- ✓ Características visuales para el desarrollo web.
- ✓ Creador gráfico de juegos para celulares.

El IDE seleccionado por el colectivo de autores para desarrollar la extensión, es el NetBeans, en su versión 6.9, por ser de código abierto, por su factibilidad de uso, además de considerarse más estable y rápido; su diseño global permite tener las herramientas que necesitan los programadores inmediatamente al alcance de sus manos. Es fácilmente integrable con la herramienta CASE “*Visual Paradigm for UML*”.

Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costes, implementación automática de parte del código con el diseño dado, compilación automática, documentación o detección de errores, entre otras. (11)

“*Visual Paradigm for UML*”: es una herramienta CASE que usa UML como lenguaje de modelado. La misma soporta, completamente, el ciclo de vida del desarrollo de software: Análisis y Diseño Orientados a Objetos, Construcción, Pruebas y Despliegue. También permite la generación automática de reportes en formato PDF y HTML, el reconocimiento de artefactos de ingeniería a partir de reconocimiento de textos, implementa una actualización automática del modelo de diseño y código permitiendo mantener la documentación de ambos modelos actualizadas con los cambios que ocurran en ambos sentidos, optimizando la descripción textual de elementos de código a partir de la descripción visual. Es capaz de importar y exportar elementos de otras herramientas CASE como Rational Rose y presenta una interfaz con increíbles facilidades a la hora de modelar los diagramas. (12)

Dentro de las principales características de la herramienta están:

- ✓ Soporte de UML versión 2.0.
- ✓ Disponibilidad de integrarse en los principales IDE.
- ✓ Soporta una gama de lenguajes en la Generación de Código e Ingeniería Inversa en Java, C++, CORBA IDL, PHP, Ada y Python.
- ✓ Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio.

- ✓ Diagramas de flujo de datos.
- ✓ Generador de informes para generación de documentación.
- ✓ Distribución automática de diagramas. Importación y exportación de ficheros.
- ✓ Editor de figuras.

"*Visual Paradigm for UML*" es la herramienta CASE que se utilizará en el modelado de esta herramienta debido a que es multiplataforma, está definida en la arquitectura del departamento, además de ser una potente herramienta CASE para uso de la comunidad de desarrolladores. Permite modelar UML, con 13 tipos diferentes de diagramas que pueden ser exportados como imágenes en formato JPG, PNG, entre otros. Proporciona a los desarrolladores una plataforma con interfaz amigable que les permite diseñar un producto con calidad de forma muy rápida. Puede ser extendido, pues presenta soporte al diseño personalizado, lo que permite incorporar nuevas formas y notaciones, mediante el uso de imágenes o íconos importados. Debido a que la herramienta no posee el proceso de administración de requisitos se hace necesario que la herramienta garantice esto, por lo que se decide extenderla.

La implementación de extensiones para aplicaciones constituye un mecanismo para la incorporación de funcionalidades que la aplicación no provee a los usuarios. Por lo general se asocian las extensiones con *plugin,* que son fragmentos de software que interactúan con el núcleo de la aplicación para proporcionar algunas funcionalidades que en la mayoría de los casos son muy específicos. "*Visual Paradigm for UML*" es una de las herramientas *CASE* que goza de gran prestigio para el modelado de software, pero, presenta inconveniente para el proceso de administración de requisitos. Sin embargo cuenta con los medios para extender funcionalidades dando soporte a las extensiones de aplicación. La aplicación provee de forma libre una interfaz de programación (API por su siglas en inglés Interfaz de Programación de Aplicaciones) permitiendo a los desarrolladores implementar y reutilizar clases e interfaces, desarrollando funciones agregadas que son útiles para el desarrollo de software.

Sistema Gestor de Base de Datos

Un Sistema Gestor de Bases de Datos (SGBD) es un conjunto de programas que brinda la posibilidad a los usuarios de crear y mantener una base de datos, por lo tanto, un SGBD es un software de propósito general que facilita el proceso de definir, construir y manipular la base de datos para diversas aplicaciones.

PostgreSQL: es un sistema de base de datos relacional perteneciente al ámbito del software libre que se destaca por su robustez, escalabilidad y cumplimiento de los estándares SQL. Cuenta con

versiones para una amplia gama de sistemas operativos, entre ellos: Linux, Windows, Mac SX, Solaris y otros más.

Características de PostgreSQL

- ✓ Soporta distintos tipos de datos.
- ✓ Incorpora una estructura de datos “array”.
- ✓ Permite la declaración de funciones propias así como la definición de disparadores.
- ✓ Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- ✓ Permite la gestión de diferentes usuarios, así como también los permisos asignados a cada uno de ellos.

Se decidió utilizar un gestor de base de datos ya que se debe tener constancia de la información que se inserta en la aplicación, con la idea principal de no tener que volver a entrar estos datos, y como una forma rápida de generar reportes de la gestión de los requisitos en cualquier momento.

Por todo lo anteriormente expuesto se decide utilizar el Sistema Gestor de Bases de Datos (SGBD) PostgreSQL 8.4 ya que es de código abierto bajo licencia BSD, es decir, sus potencialidades están en constante perfeccionamiento, lo que permite su uso y distribución sin costo; además, continuamente se elimina y mejora cualquier hueco de seguridad que pueda aparecer debido a que sus usuarios pueden acceder a su código y modificarlo a sus necesidades. En el caso que se esté trabajando con sistemas de alta seguridad, la integridad referencial de los datos es muy buena, se pueden crear funciones complejas para validar datos. Es también utilizado porque es un SGBD potente y multiplataforma.

1.5 Elementos arquitectónicos de la extensión

Una extensión o comúnmente llamada plugin es un módulo de software o de hardware que añade características o funcionalidades específicas a una herramienta más grande, en nuestros días es muy común ver como las grandes aplicaciones sufren cambios por parte de los programadores ya que las técnicas de trabajo van evolucionando y se necesita de nuevas alternativas que las mismas no pueden brindar. “*Visual Paradigm for UML*” es una excelente herramienta para el modelado UML y el trabajo de analistas e ingenieros de software, la misma cuenta con los medios para extenderse a gran escala. Los programadores pueden implementar, reutilizar sus clases e interfaces ya que provee de una interfaz de programación API que le permite esta integración.

Para la implementación de la extensión hay que tener en cuenta algunos puntos importantes:

Integración: la herramienta “Visual Paradigm for UML” permite la integración de poder ser extendida mediante la biblioteca openapi.jar, es necesario tener en cuenta la estructura que sigue la extensión para su implementación, la misma presenta un paquete general con 3 paquetes asociados, el primero con el nombre de plugin el cual representa las configuraciones de la extensión, el segundo paquete contiene las acciones definidas para realizar y el último paquete presenta los diálogos o formularios, se debe tener buen dominio acerca de la integración de todos estos paquetes ya que todos están relacionados entre sí.

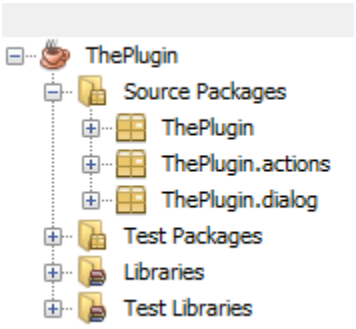


Ilustración 4 Estructura general de paquetes

Para el primer paquete es necesario conocer que está conformado por un conjunto de clases que permite cargar y configurar la extensión, las mismas son Plugin.xml y Plugin.java.

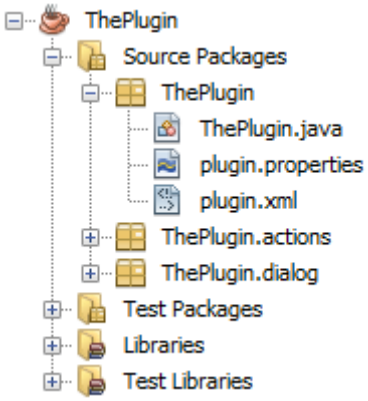


Ilustración 5 Estructura del primer paquete

El segundo paquete contiene las acciones de la extensión definidas a nivel de herramientas y de contexto. Dichas clases en dependencia de la acción, implementa interfaces asociadas a las acciones, VPActionController y VPContextAction. Ambas clases definen el performAction el cual permite ejecutar acciones referentes al evento onClick de la acción.

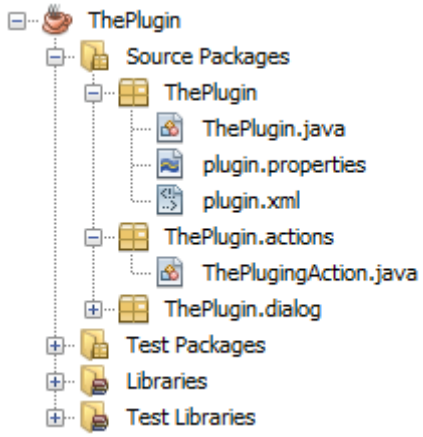


Ilustración 6 Estructura del segundo paquete

El tercer paquete contiene los diálogos que se desean mostrar, los diálogos se muestran mediante el método performAction asociado a la clase de acción correspondiente al dialogo.

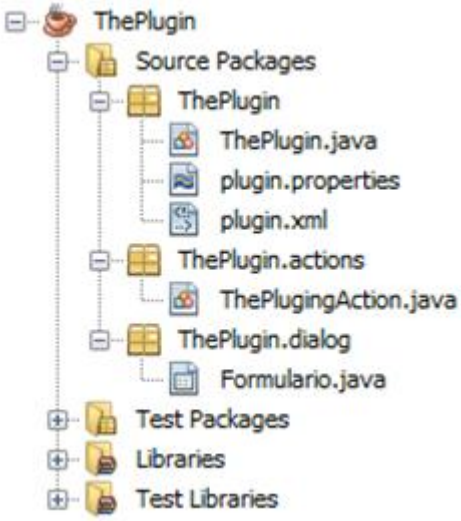


Ilustración 7 Estructura del tercer paquete

El segundo y último paso es la integración de la extensión según lo que propone "*Visual Paradigm for UML*"

Integración: "Visual Paradigm for UML" propone crear una carpeta en su directorio de instalación con el nombre de plugins, dentro de la cual estará la carpeta de la extensión con la siguiente estructura.

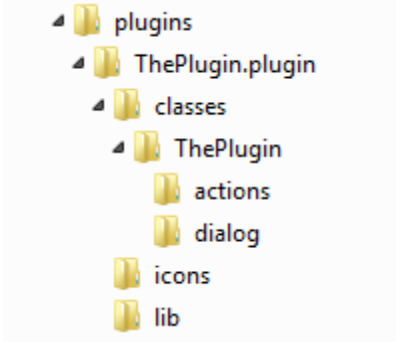


Ilustración 8 Estructura de despliegue

Si se observa la estructura de implementación de la extensión, es diferente a la de integración con la herramienta, lo que dificulta el proceso de pruebas al integrar la extensión implementada para "*Visual Paradigm for UML*". La solución a este inconveniente es utilizar una herramienta de despliegue de extensiones que facilite la integración del mismo con "*Visual Paradigm for UML*", pasando los valores a la herramienta como es el nombre de la extensión, el directorio de origen de la implementación en el IDE y la dirección donde será desplegada la extensión. (13)

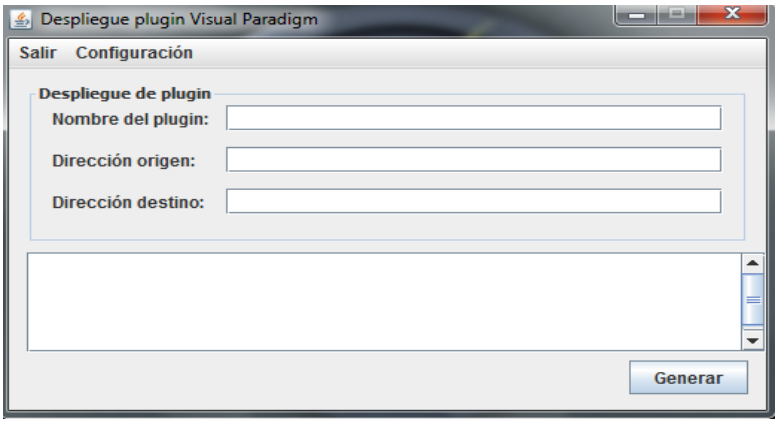


Ilustración 9 Herramienta para despliegue de la extensión

1.6 Conclusiones parciales del capítulo

Luego de haber realizado un estudio sobre la Ingeniería de Requisitos, sus procesos y herramientas informáticas, se definió como lenguaje de programación Java, como IDE de desarrollo NetBeans 6.9, OpenUP como la metodología a utilizar, "*Visual Paradigm for UML*" 8.0 como herramienta CASE para el modelado, como gestor de base de datos PostgreSQL 8.4; estas herramientas con el fin de crear un sistema que cumpla con la calidad requerida y apoye al desarrollo de la extensión para garantizar la administración de requisitos.

Análisis y diseño de la extensión

En el presente capítulo se describen los conceptos más importantes en el entorno donde estará el sistema. Se identifican los requisitos funcionales y no funcionales que se deben tener en cuenta para la implementación de la extensión. Se muestran los diferentes tipos de diagramas que representan la lógica y funcionamiento del sistema, con el objetivo de detallar las actividades para un mejor entendimiento de todo el proceso que se lleva a cabo en la administración de requisitos y, también se muestran los patrones de diseño que se aplican en la extensión.

2.1 Modelo de Dominio

Un modelo del dominio captura los tipos de objetos más importantes que existen, o los eventos que suceden en el entorno donde estará el sistema, se identifican conceptos, se definen estos conceptos y se unen o relacionan en un diagrama de clases UML. El objetivo fundamental de este modelo es comprender y describir los conceptos más importantes dentro del contexto del sistema. El modelo de dominio es una representación visual del entorno real del proyecto, proporciona una perspectiva conceptual (objetos del dominio o clases conceptuales, asociaciones entre clases conceptuales y atributos de las clases conceptuales), la información contenida puede ser expresada en forma de texto plano, este tipo de diagrama disminuye la brecha de representación entre cómo ven los clientes el problema y la representación en software de la solución, usando modelado Orientado por Objetos. (14) (15)

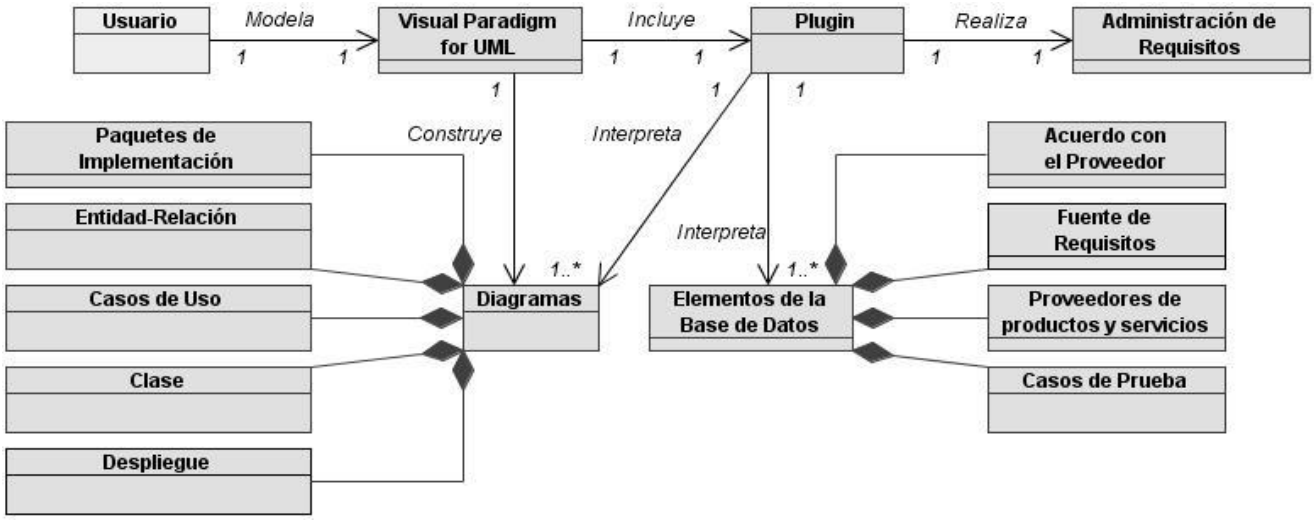


Ilustración 10 Modelo de Dominio

Definición de conceptos del Modelo de Dominio

Usuario: representa a los usuarios que accederán a trabajar con la extensión desde el "*Visual Paradigm for UML*".

"*Visual Paradigm for UML*": representa la herramienta CASE donde estará enmarcada la extensión y la encargada de modelar y contener los diagramas para la posterior interpretación.

Diagramas: representación gráfica en la que se muestran las relaciones entre las diferentes partes de un conjunto o sistema o los cambios de un determinado fenómeno, las cuales van a ser interpretadas por la extensión.

Paquetes de Implementación: representa el tipo de diagrama que su nombre indica, este tipo de diagrama se encuentra en la herramienta "*Visual Paradigm for UML*" y la extensión interpreta todos sus componentes en forma de objetos.

Entidad - Relación: representa el tipo de diagrama que su nombre indica, este tipo de diagrama se encuentra en la herramienta "*Visual Paradigm for UML*" y la extensión interpreta todos sus componentes en forma de objetos, es una percepción del mundo real representado por entidades, que son objetos que existen y son importantes en el problema a resolver, que incluyen atributos y se relacionan entre ellos. Es el punto de partida, para la generación del diagrama de clases a través de transformación de modelos.

Caso de Uso: un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participarán en un caso de uso se denominan actores. En el contexto de ingeniería del software, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. La extensión interpreta todos sus componentes en forma de objetos.

Despliegue: representa el tipo de diagrama que su nombre indica, este tipo de diagrama se encuentra en la herramienta "*Visual Paradigm for UML*" y la extensión interpreta todos sus componentes en forma de objetos, se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes.

Diagrama de Clase: representa el tipo de diagrama que su nombre indica, este tipo de diagrama se encuentra en la herramienta "*Visual Paradigm for UML*" y la extensión interpreta todos sus componentes en forma de objetos, es un diagrama representado por clases, que son objetos

relacionados entre sí, que contienen atributos y operaciones. Representan la fuente de obtención de código a partir de la realización del diagrama entidad relación antes mencionado.

Elementos de la Base de Datos: representa de forma general a todos los tipos de documentos que tienen relación directa con los requisitos con los cuales la extensión va a trabajar y realizar una trazabilidad entre cada uno de ellos.

Acuerdo con el Proveedor: representa el documento que su nombre indica, este elemento se encuentra en la Base de Datos.

Fuente de Requisitos: representa el documento que su nombre indica, este elemento se encuentra en la Base de Datos. Indica la procedencia del requisito.

Proveedores de Productos y Servicios: representa el documento que su nombre indica, este elemento se encuentra en la Base de Datos.

Casos de Prueba: representa el documento que su nombre indica, este elemento se encuentra en la Base de Datos.

Extensión (Plugin): módulo que puede ser de Hardware o de Software que agrega características o funcionalidades a una aplicación más grande. Representa la extensión en su totalidad, y es la encargada de interpretar los diferentes tipos de diagramas que se construyen en la herramienta "*Visual Paradigm for UML*" y los elementos de la Base de Datos para posteriormente realizar la trazabilidad con los requisitos que el usuario inicial en la extensión, con todos los datos la extensión es capaz de generar matrices de trazabilidad que permitan una mejor administración de los requisitos que se están estudiando por el usuario, siguiendo la guía que plantea el proceso de mejora de la universidad.

Administración de Requisitos: representa el resultado obtenido por la extensión en su totalidad, en resumen es el resultado que el usuario espera obtener, mediante el proceso anteriormente explicado en las clases pertenecientes al modelo de dominio.

Para una mayor información ver: (*Expediente de Proyecto*).

2.2 Especificación de los Requisitos del Sistema

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Definen las funciones que el sistema será capaz de realizar. Expresan la naturaleza del funcionamiento del sistema, cómo interacciona el sistema con su entorno y cuáles van a ser su estado y funcionamiento. (16)

Se identificaron los siguientes requisitos funcionales.

RF1_Establecer conexión con la base de datos.

Descripción: el usuario ingresa los datos correspondiente para entrar a la aplicación.

Entrada: datos de la conexión (usuario, contraseña, servidor, host, puerto).

Salida: mensaje de confirmación del sistema, indicando el estado de la conexión.

RF2_Buscar proyecto.

Descripción: se realiza una búsqueda de los proyectos.

Entrada: se realiza una búsqueda automática de los proyectos.

Salida: listado de los proyectos encontrados.

RF3_Insertar proyecto.

Descripción: se realiza la inserción de los proyectos.

Entrada: se ingresan los criterios de búsqueda (nombre del proyecto, centro y descripción).

Salida: mensaje de confirmación de la inserción.

RF4_ Eliminar proyecto.

Descripción: se selecciona cual proyecto se desea eliminar de la lista de todos los proyectos.

Entrada: la lista con los posibles proyectos a eliminar.

Salida: mensaje de confirmación de la eliminación.

RF5_Actualizar proyecto.

Descripción: se actualizan los datos del proyecto.

Entrada: campos para la actualización (nombre del proyecto, centro, descripción).

Salida: mensaje de confirmación de la actualización.

RF6_Buscar módulo.

Descripción: se realiza la búsqueda de los módulos existentes en la aplicación.

Entrada: se realiza una búsqueda automática de los módulos.

Salida: listado de los módulos encontrados.

RF7_Insertar módulo.

Descripción: se ingresan los datos para la inserción.

Entrada: se ingresan los datos (nombre del módulo, proyecto al que pertenece).

Salida: mensaje de confirmación de la inserción.

RF8_Eliminar módulo.

Descripción: se selecciona cual módulo se desea eliminar de la lista de todos los módulos.

Entrada: la lista con los posibles módulos a eliminar.

Salida: mensaje de confirmación de la eliminación.

RF9_Actualizar módulo.

Descripción: se actualizan los datos del módulo.

Entrada: campos para la actualización (nombre del módulo, proyecto al que pertenece).

Salida: mensaje de confirmación de la actualización.

RF10_Buscar elemento.

Descripción: se realiza la búsqueda de los elementos existentes en la base de datos.

Entrada: se realiza una búsqueda automática de los elementos de la Base de Datos.

Salida: listado de los elementos encontrados.

RF11_Insertar elemento.

Descripción: se ingresan los datos para la inserción.

Entrada: se ingresan los datos (nombre).

Salida: mensaje de confirmación de la inserción.

RF12_Eliminar elemento.

Descripción: se selecciona cual elemento se desea eliminar de la lista de todos los elementos.

Entrada: la lista con los posibles elementos a eliminar.

Salida: mensaje de confirmación de la eliminación.

RF13_Actualizar elemento.

Descripción: se actualizan los datos del elemento.

Entrada: campos para la actualización (nombre).

Salida: mensaje de confirmación de la actualización.

RF14_Buscar requisito.

Descripción: se realiza la búsqueda de los requisitos existentes en la base de datos.

Entrada: se realiza una búsqueda automática de los requisitos.

Salida: listado de los requisitos encontrados.

RF15_Insertar requisito.

Descripción: se ingresan los datos para la inserción.

Entrada: se ingresan los datos (nombre, módulo, proyecto).

Salida: mensaje de confirmación de la inserción.

RF16_Eliminar requisitos.

Descripción: se selecciona cual requisitos se desea eliminar de la lista de todos los requisitos.

Entrada: la lista con los posibles requisitos a eliminar.

Salida: mensaje de confirmación de la eliminación.

RF17_Actualizar requisitos.

Descripción: se actualizan los datos de los requisitos.

Entrada: campos para la actualización (nombre, módulo, proyecto).

Salida: mensaje de confirmación de la actualización.

RF18_ Generar Matriz de Trazabilidad.

Descripción: se seleccionan los elementos para tracear.

Entrada: listado con los elementos.

Salida: matriz de Trazabilidad.

RF19_ Interpretar diagrama.

Descripción: se seleccionan los diagramas existentes.

Entrada: listado con los diagramas.

Salida: listado de los elementos para realizar la trazabilidad.

RF20_Generar reportes.

Descripción: se seleccionará el tipo de reporte.

Entrada: opciones de reportes disponibles.

Salida: reporte en el formato seleccionado con la información correspondiente.

Para una mayor información ver: (*Expediente de Proyecto*).

Se identificaron los siguientes requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general los requisitos no funcionales son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, es decir, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades o propiedades debe tener. (16)

Por lo anteriormente expuesto se definen los siguientes requisitos no funcionales.

Requisitos de Software

- ✓ La herramienta *"Visual Paradigm for UML" 8.0*.
- ✓ Se debe tener instalado el JRE 6.0 o superior (Java Runtime Environment).
- ✓ Se debe tener instalado el Sistema Gestor de Base Datos (PostgreSQL 8.4).

Requisitos de Hardware

- ✓ Intel Pentium III, Compatible con procesador a 1.0 GHz o superior.
- ✓ Mínimo 512 MB de RAM (Random Access Memory, por sus siglas en inglés), pero se recomienda 1GB.
- ✓ Un mínimo de 1GB de espacio en disco.

Restricciones del diseño y la implementación

- ✓ Se debe utilizar la herramienta *"Visual Paradigm for UML"* en su suite 5.0 y el IDE NetBeans 6.9.
- ✓ El lenguaje de programación que será usado para la implementación es Java siguiendo el paradigma de la Programación Orientada a Objeto y su compatibilidad con otros sistemas operativos.

Requisitos de Usabilidad

- ✓ Facilidad de uso por parte de los usuarios: La interfaz debe ser lo más descriptiva posible, permitiendo que las operaciones a realizar por los usuarios estén bien descritas, de manera que se puedan entender claramente.
- ✓ La interfaz debe tener mensajes contextuales asociados a los objetos.

Requisitos de Portabilidad

- ✓ La extensión una vez integrado a la herramienta "*Visual Paradigm for UML*" podrá ser instalado y disponer del mismo en diferentes sistemas operativos por ser "*Visual Paradigm for UML*" una herramienta multiplataforma.

Seguridad

- ✓ El usuario que desee usar la extensión deberá ser autenticado previamente para tener acceso a los datos almacenados en la Base de Datos.

Disponibilidad

- ✓ La extensión debe estar disponible en la herramienta "*Visual Paradigm for UML*" siempre que se desee trabajar, de la misma manera con la Base de Datos.

2.3 Modelo de Casos de Uso del Sistema

El modelo de casos de uso del sistema representa las relaciones existentes entre actores y casos de uso. Un modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema, representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un caso de uso es una unidad de trabajo significativo. Cada caso de uso tiene una descripción que especifica la funcionalidad que se incorporará al sistema propuesto. Un caso de uso puede 'incluir' la funcionalidad de otro caso de uso o puede 'extender' a otro, con su propio comportamiento. (17) (15)

Actor del sistema

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participarán en un caso de uso se denominan actores. En el contexto de ingeniería del software, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. (18)

Nombre del Actor	Descripción
Usuario	Es el encargado de realizar la gestión de los elementos dentro de la extensión, y de esta forma poder realizar una correcta administración de los requisitos.

Tabla 2 Descripción del Actor del Sistema

A continuación se muestra el diagrama de casos de uso del sistema el mismo posee 8 casos de uso, los cuales fueron agrupados utilizando patrones de casos de uso sobre los requisitos funcionales. (19)

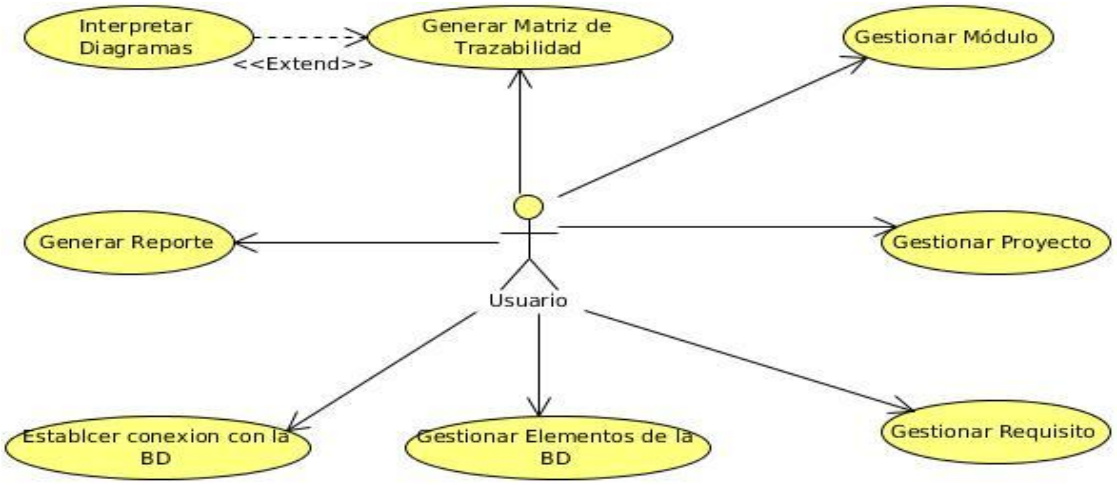


Ilustración 11 Diagrama de Casos de Uso del Sistema

CUS 1. Establecer conexión con la Base de Datos: el usuario realiza la conexión a la Base de Datos ingresando los datos correspondientes para la misma.

CUS 2. Gestionar Proyectos: el usuario realiza la gestión de los proyectos, con el objetivo de llevar un mejor control de los requisitos.

CUS 3. Gestionar Módulos: el usuario realiza la gestión de los módulos, con el objetivo de llevar un mejor control de los requisitos.

CUS 4. Gestionar Elementos: el usuario realiza la gestión de los elementos, con el objetivo de llevar un mejor control de los requisitos.

CUS 5. Gestionar Requisitos: el usuario realiza la gestión de los requisitos, con el objetivo de llevar un mejor control de los requisitos.

CUS 6. Generar Matriz de Trazabilidad: el usuario genera la matriz de trazabilidad de los requisitos.

CUS 7. Interpretar Diagramas: la extensión interpreta los diagramas en el "*Visual Paradigm for UML*" en forma de objetos para realizar la trazabilidad.

CUS 8. Generar Reportes: el usuario genera reportes con la información deseada o con las matrices correspondientes.

Para una mayor información ver: (*Expediente de Proyecto*).

Matriz de Trazabilidad

Es la representación gráfica de las relaciones entre dos o más productos del proceso de desarrollo, para representar la relación entre los requisitos y otros elementos del sistema. (20)

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	RF10	RF11	RF12	RF13	RF14	RF15	RF16	RF17	RF18	RF19	RF20
CUS-1	X																			
CUS-2		X	X	X	X															
CUS-3						X	X	X	X											
CUS-4										X	X	X	X							
CUS-5														X	X	X	X			
CUS-6																		X		
CUS-7																			X	
CUS-8																				X

Tabla 3 Matriz de Trazabilidad CUS – Requisitos

En este caso para lograr un correcto cumplimiento de las funcionalidades planteadas, se comprueba mediante la matriz de trazabilidad que los requisitos cubran los casos de uso.

2.4 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización de casos de uso, y sirve como entrada al su código fuente. El modelo de diseño se utiliza como parte esencial para las actividades en ejecución y prueba, que se basa en el análisis y los requisitos de la arquitectura del sistema. Representa los componentes de aplicación y determina su colocación adecuada y el uso dentro de la arquitectura en general del sistema. (15)

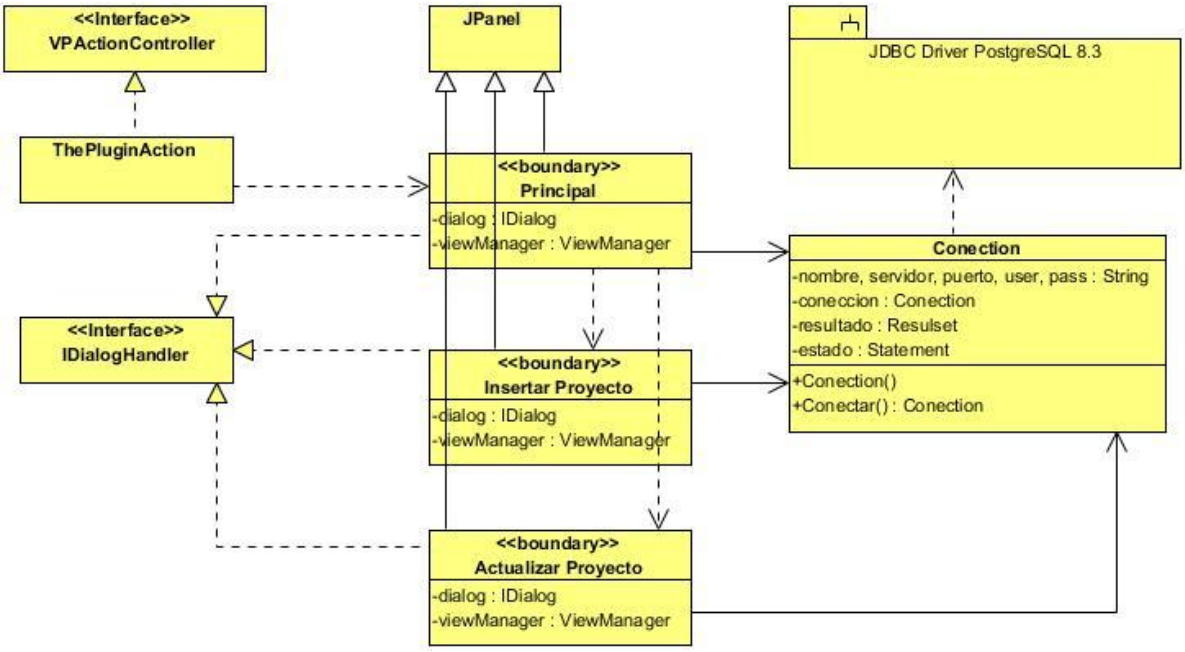


Ilustración 12 Diagrama de Clase del Diseño CUS Gestionar Proyecto

ThePluginAction: es la clase que permite la entrada al plugin al acceder desde el "*Visual Paradigm for UML*".

VPActionController: es la clase interfaz controladora del plugin que propone el api de acceso al "Visual Paradigm for UML", para el acceso al mismo.

Conection: es la clase donde se encuentran los métodos genéricos de conexión y acceso a la Base de Datos.

JDBC Driver PostgreSQL 8.3: es el subsistema utilizado por el plugin para lograr realizar la conexión.

IDialogHandler: es la clase interfaz que implementan las clases interfaces de usuario para ser mostradas en el "Visual Paradigm for UML".

JPanel: es la clase del lenguaje java que permite tener componentes visuales para mostrar al usuario, todas las clases interfaces deben extender de esta para poder ser mostradas.

Principal: es la clase interfaz principal de la cual parten todas las demás interfaces del plugin.

Insertar Proyecto: es la clase interfaz encargada de garantizar la inserción de un proyecto.

Actualizar Proyecto: es la clase interfaz encargada de garantizar la actualización de un proyecto.

Para una mayor información ver: (*Expediente de Proyecto*).

Patrones Utilizados

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Muchos son los autores que definen que es un patrón de diseño, pero todas convergen en la definición dada por Christopher Alexander y la cual se tomó como punto de partida:

"… cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal forma que puede usar esa solución un millón de veces más, sin haberlo hecho de la misma forma dos veces". (21)

- **Singleton**

Descripción:

Es de tipo creacional, a nivel de objetos. Su propósito es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. El acceso a la "Instancia Única" es controlado. Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario". Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable). (22)

¿Cuándo utilizarlo?

Se utiliza cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes.

¿Cómo funciona?

El patrón Singleton o Solitario es implementado por la clase PluginController.java que permite la creación de objetos a través de instancias, manteniendo la consistencia entre los objetos.
En el siguiente fragmento de código se pone de manifiesto el patrón Singleton, haciendo una llamada a una instancia solo si no está creada la misma.

```
public void performAction(VPAction action) {

    if (!ThePluginController.getInstance().hasConection()) {
        IConection obj  = new IConection(viewManager);
        this.viewManager.showDialog(obj);
    }
    Principal principal = new Principal(viewManager);
    viewManager.showDialog(principal);
}
```

Ilustración 13 Representación de llamada. Patrón Singleton

- **Experto**

Descripción:

Lo que plantea este patrón es asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para cumplir con la responsabilidad. El problema que resuelve el patrón experto está referido al principio más básico mediante el cual las responsabilidades son asignadas en el diseño orientado a objetos.

El modelo de clases puede definir docenas o cientos de clases de software, y una aplicación puede definir cientos o miles de responsabilidades a ser cumplidas. Durante el diseño orientado a objetos, cuando las interacciones entre objetos son definidas, se realizan escogencias acerca de la asignación de responsabilidades a las clases. Haciéndolas bien, los sistemas tienden a ser fáciles de entender, mantener y extender, y hay una oportunidad de rehusar los componentes en aplicaciones futuras.

Aplicabilidad:

El experto es usado más que cualquier otro patrón en la asignación de responsabilidades, es un principio continuamente en el diseño orientado a objetos. Experto no significa una obscura idea, sino que expresa la "intuición" común mediante el cual los objetos hace cosas relacionadas con la información que ellos tienen.

Beneficios:

La encapsulación es mantenida, desde que los objetos usan sus propias informaciones para realizar tareas. Esto permite poco acoplamiento, lo cual conduce a sistemas más robustos y de mantenimiento mucho más fácil.

El comportamiento está distribuido a lo largo de clases que tienen la información requerida, así se alienta una clase "peso ligero" de definiciones que son fáciles de entender y mantener. La alta cohesión también es soportada. (23)

Ilustración 14 Clase Interfaz Generar Reporte. Patrón Experto

2.5 Diagramas de Interacción

Un diagrama de interacción se entiende como la representación de secuencias que componen un sistema, para facilitar el entendimiento de información y la relación que mantiene entre las diferentes partes, modelan el comportamiento dinámico del sistema; el flujo de control en una operación, describe la interacción entre objetos; los objetos interactúan a través de mensajes para cumplir ciertas tareas, proveen un comportamiento y típicamente implementan un caso de uso. Existen dos tipos de interacción en UML.

- ✓ Diagramas de Secuencia (Dimensión temporal)
- ✓ Diagramas de Colaboración (Dimensión estructural)

Diagramas de Secuencias

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista de negocio del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos. (24) (15)

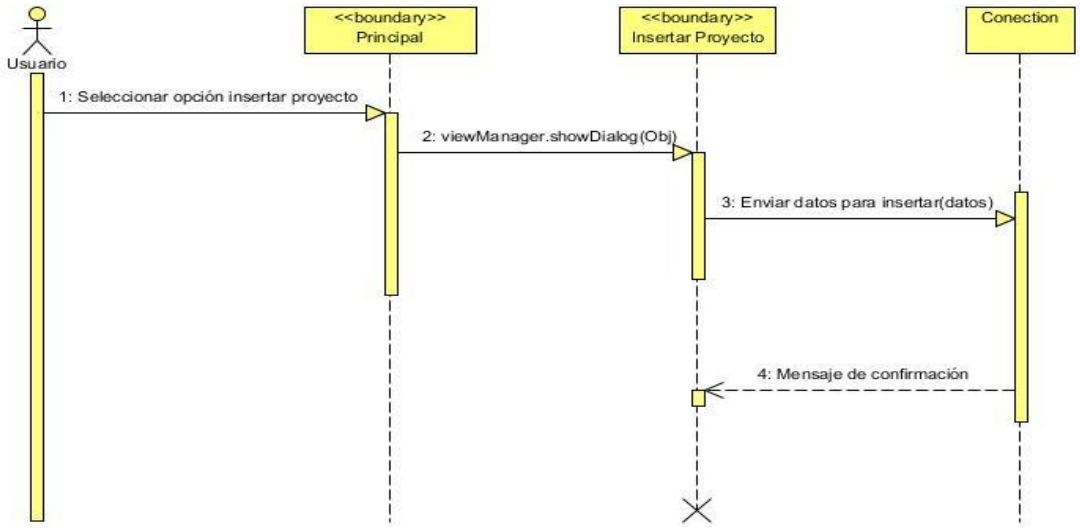


Ilustración 15 Diagrama de Secuencia – Escenario: Insertar Proyecto

Usuario: interactúa con la extensión, en este caso con la interfaz Gestionar Proyecto, interfaz que se selecciona desde la principal.

Principal: es la clase interfaz de usuario principal de la cual parten todas las demás interfaces del plugin.

Insertar Proyecto: es la clase interfaz de usuario capaz de insertar proyectos a la extensión enviando los datos a la Base de Datos y verificándolos.

Conection: es la clase donde se encuentran los métodos genéricos de conexión y acceso a la Base de Datos.

Para una mayor información ver: (*Expediente de Proyecto*).

2.6 Modelo de Datos Relacional

El modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postuladas sus bases en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos.

Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, esto es, pensando en cada relación como si fuese una tabla que está compuesta por registros (cada fila de la tabla sería un registro o tupla), y columnas (también llamadas campos). (25) (15)

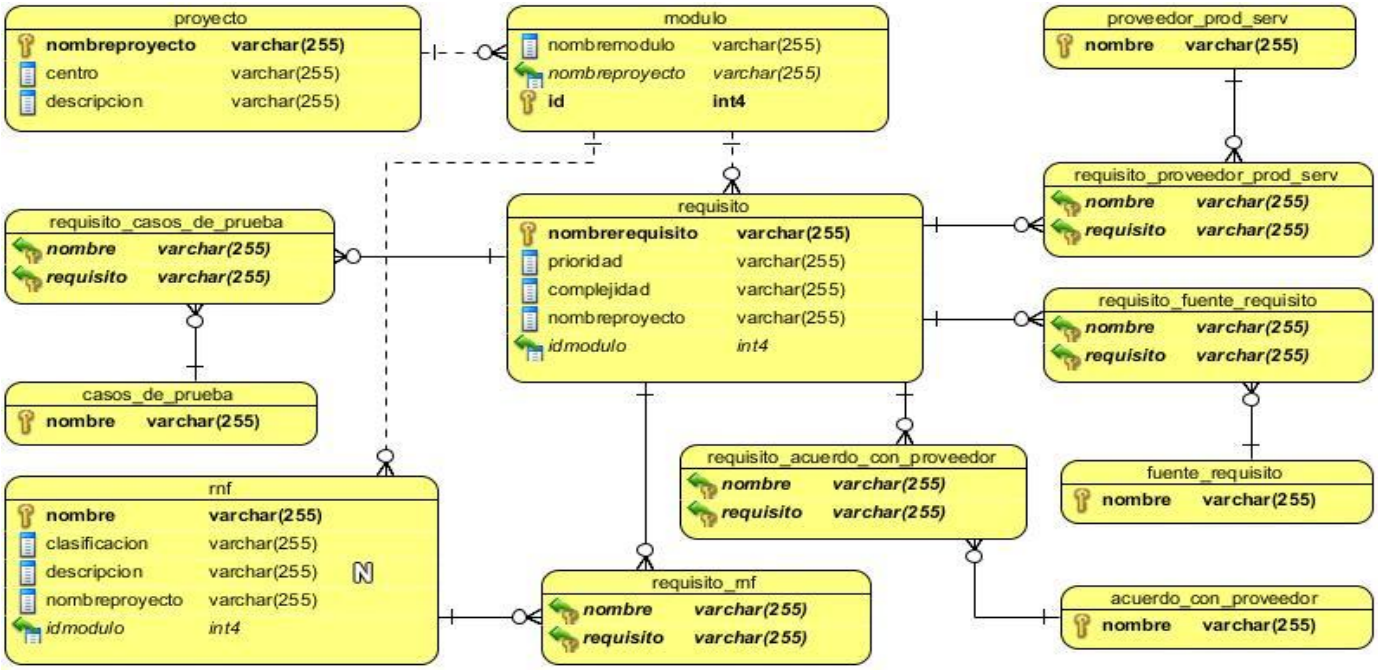


Ilustración 16 Modelo de Datos

Nombre: proyecto		
Descripción: es la tabla de la Base de Datos en la cual se almacenan los proyectos insertados.		
Relación con la tabla: modulo		
Tipo de relación: Uno – Mucho		
Atributo	Tipo	Descripción
nombreproyecto	Varchar (255)	Nombre que tendrá el proyecto
centro	Varchar (255)	Nombre del centro al que pertenece
descripción	Varchar (255)	Es el campo opcional para describir al proyecto

Tabla 4: tabla de la Base de Datos: proyecto

Nombre: modulo		
Descripción: es la tabla de la Base de Datos en la cual se almacenan los módulos insertados.		
Relación con la tabla: requisito, rnf		
Tipo de relación: Uno – Mucho		
Atributo	Tipo	Descripción
id	int	Identificador del módulo
nombremodulo	Varchar (255)	Nombre del módulo
nombreproyecto	Varchar (255)	Nombre del proyecto al que pertenece

Tabla 5: tabla de la Base de Datos: modulo

Nombre: requisito		
Descripción: es la tabla de la Base de Datos en la cual se almacenan los requisitos insertados.		
Relación con la tabla: Elementos (casos_de_prueba, fuente_requisito, acuerdo_con_proveedor, proveedor_prod_serv, rnf)		
Tipo de relación: Mucho – Mucho		
Atributo	Tipo	Descripción
nombrerequisito	Varchar (255)	Nombre que tendrá el requisito
prioridad	Varchar (255)	Tipo de prioridad del requisito
complejjidad	Varchar (255)	Complejidad del requisito
nombreproyecto	Varchar (255)	Nombre al que pertenece
idmodulo	int	Identificador del módulo al que pertenece

Tabla 6: tabla de la Base de Datos: requisito

La tabla rnf (requisitos no funcionales) de la Base de Datos presentada en el modelo de datos, mantiene la misma estructura que la tabla anterior (requisito). A continuación se presenta la muestra

de una de las tablas pertenecientes a los elementos a tracear y de esta manera ver cómo se comportan en los otros casos. Cada una de estas tablas correspondientes a los elementos y mantienen una relación Mucho – Mucho con la tabla requisitos generándose una nueva tabla donde se almacena la relación que existe entre los requisitos y cada elemento. Para una mayor información ver: (*Expediente de Proyecto*).

Nombre: casos_de_prueba		
Descripción: es la tabla de la Base de Datos en la cual se almacenan los elementos pertenecientes a los casos de prueba.		
Relación con la tabla: requisito		
Tipo de relación: Mucho - Mucho		
Atributo	Tipo	Descripción
nombre	Varchar (255)	Nombre del elemento

Tabla 7: tabla de la Base de datos: casos_de_prueba

2.7 Conclusiones parciales del capítulo

En este capítulo se presentaron los requisitos funcionales y no funcionales que la extensión presenta, como se definió por el grupo de trabajo, así como el modelo del dominio, se realizó el diagrama de casos de uso del sistema se evidencia la relación del actor y los casos de uso. Para tener una mejor organización se realizaron los diagramas de clases del diseño para cada caso de uso y sus respectivos diagramas de interacción para ver el comportamiento. También fue realizado el modelo de datos relacional, donde se especifican las clases persistentes de la Base de Datos. Finalmente se presentaron los patrones de diseño que se utilizarán en la extensión.

Implementación y prueba de la extensión

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

3.1 Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables, etc. Describe también cómo se organizan los componentes de acuerdo a los mecanismos de estructuración y modularización disponibles en el entorno de implementación y lenguaje o lenguajes de implementación empleados, y cómo dependen los componentes unos de otros. (15)

Un diagrama de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo**,** se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos. Consiste en:

- ✓ **Nodos:** elementos de procesamiento con al menos un procesador, memoria, y posiblemente otros dispositivos.
- ✓ **Dispositivos:** nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.
- ✓ **Conectores:** expresa el tipo de conector o protocolo utilizado entre el resto de los elementos del modelo.

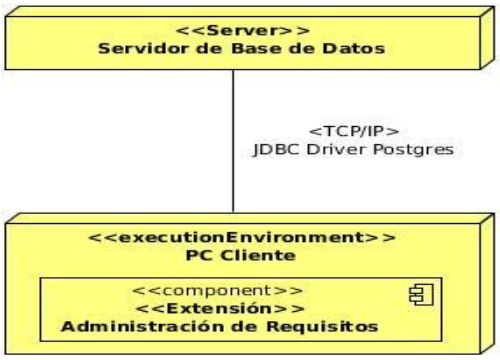


Ilustración 17 Diagrama de Despliegue de la extensión

Diagramas de Componentes

El diagrama de componentes es usado para estructurar el modelo de implementación, describe los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Un componente representa un elemento físico que forma parte del sistema, se puede representar por nodos y sus operaciones solo se pueden alcanzar a través de interfaces. (26) (15)

Existen diferentes tipos de componentes, como son:

- ✓ **Executable:** especifica un componente que se puede ejecutar en un nodo.
- ✓ **Library:** especifica una biblioteca de objetos estática o dinámica.
- ✓ **Table:** especifica un componente que representa una tabla de una base de datos.
- ✓ **File:** especifica un componente que representa un documento que contiene código fuente o datos.
- ✓ **Document:** especifica un componente que representa un documento.

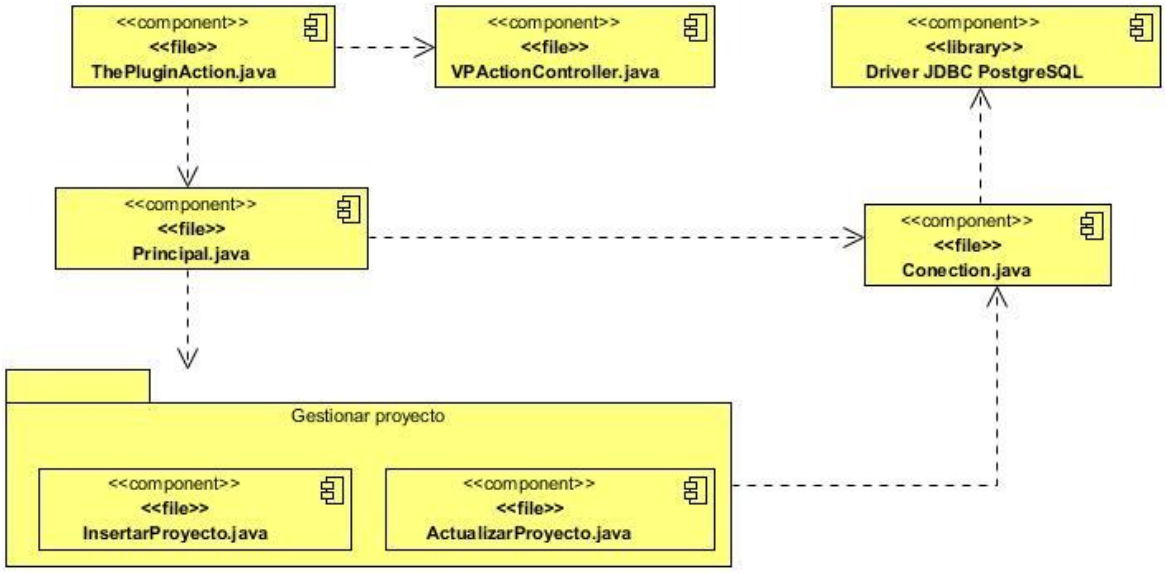


Ilustración 18 Diagrama de Componente - CU Gestionar Proyecto

Principal.java: es la clase interfaz principal de la cual parten todas las demás interfaces de la extensión.

ThePluginAction: es la clase que permite la entrada al plugin al acceder desde el "*Visual Paradigm for UML*".

VPActionController: es la clase interfaz controladora del plugin que propone el api de acceso al "Visual Paradigm for UML", para el acceso al mismo.

Gestionar Proyecto: es el paquete de componentes de interfaces donde se gestionan todos los proyectos de la extensión (Insertar, Eliminar, Buscar y Actualizar).

Conection.java: es la clase donde se encuentran los métodos genéricos de conexión y acceso a la Base de Datos.

JDBC Driver PostgreSQL 8.3: es el subsistema utilizado por la extensión para lograr realizar la conexión.

Para una mayor información ver: (*Expediente de Proyecto*).

El diagrama de componentes presentado anteriormente describe cada uno de los componentes asociados al diseño de clases propuesto para el caso de uso Gestionar Proyecto de la extensión para la herramienta, así como la relación de dependencia entre los componentes que la integran.

3.2 Código Fuente

El código fuente de un programa informático (o software) es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

El código fuente de un programa está escrito por un programador en algún lenguaje de programación, pero en este primer estado no es directamente ejecutable por la computadora, sino que debe ser traducido a otro lenguaje (el lenguaje máquina o código objeto) que sí pueda ser ejecutado por el hardware de la computadora. Para esta traducción se usan los llamados compiladores, ensambladores, intérpretes y otros sistemas de traducción. (27)

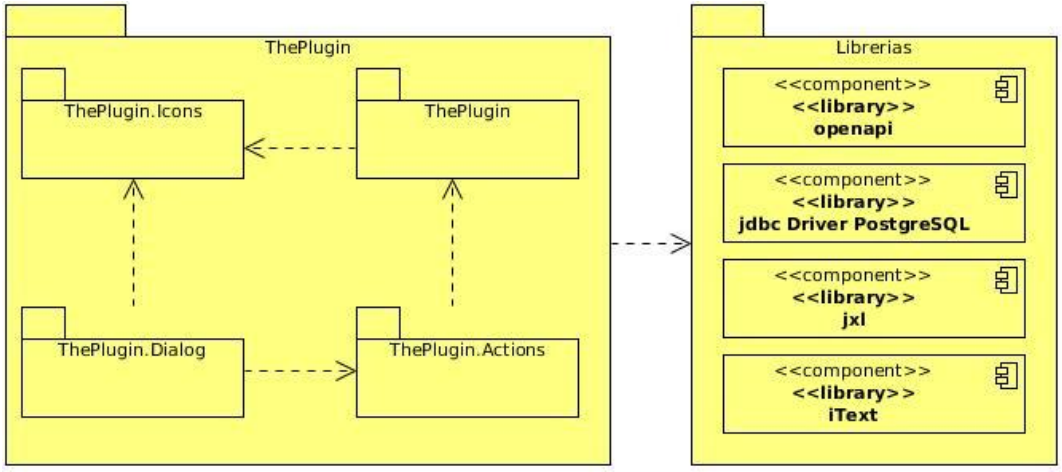


Ilustración 19 Diagrama de componente de la extensión

ThePlugin: los ficheros contenidos en este paquete pertenecen a la configuración de la extensión.

ThePlugin.Actions: los ficheros contenidos en este paquete pertenecen a las acciones que se realizan a lo largo de la extensión.

ThePlugin.Dialog: los ficheros contenidos en este paquete pertenecen a las interfaces correspondientes de las acciones realizadas en la extensión.

ThePlugin.Icons: los ficheros contenidos en este paquete pertenecen a los íconos utilizados en el diseño de la extensión.

Librerías: Los ficheros contenidos en este paquete conforman las bibliotecas utilizadas en la extensión.

- ✓ **Openapi:** biblioteca que permite la integración con "*Visual Paradigm for UML*".
- ✓ **JDBC Driver PostgreSQL:** biblioteca que permite la conexión a la Base de Datos.

- ✓ **jxl:** biblioteca que permite la creación de ficheros en formato Excel.
- ✓ **iText:** bilbioteca que permite la creación de ficheros en formato PDF.

Para una mayor información ver: (*Expediente de Proyecto*).

Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es, establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. (28)

Una declaración por línea

Se recomienda el uso de una declaración por línea, promoviendo así el uso de comentarios.
Ejemplo:

```
private IDialog          dialog;
private Conection        con;
private ViewManager      manager;
private Object[][]       crearXLS;
private Principal        principal;
```

Ilustración 20 Estándar - Declaraciones por línea

Espacios en blanco

Las líneas y espacios en blanco mejoran la legibilidad del código permitiendo identificar las secciones de código relacionadas lógicamente.

Se utilizarán espacios en blanco en los siguientes casos: entre una palabra clave y un paréntesis. Esto permite que se distingan las llamadas a métodos de las palabras clave.
Ejemplo:

```
while (rs.next())
{
    String nombre = rs.getString("nombre");
    data[i][0]=nombre;
    i++;
}
```

Ilustración 21 Estándar - Espacios en Blanco

Métodos

Los métodos deben ser verbos escritos en minúsculas. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúsculas.
Ejemplo:

```
public Connection getCon() {
    return con;
}
```

Ilustración 22 Estándar - Nombre de Métodos

3.3 Pruebas del Software

El único instrumento adecuado para determinar el status de la calidad de un producto software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. En las pruebas se usan casos de prueba, especificados de forma estructurada mediante Técnicas de prueba. El proceso de pruebas, sus objetivos y los métodos y técnicas usados se describen en el plan de prueba.

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Las pruebas son un proceso que se enfoca sobre la lógica interna del software y las funciones externas, es un proceso de ejecución de un programa con la intención de descubrir un error. Una prueba tiene éxito si descubre un error no detectado hasta entonces. Además, esta etapa implica: (29)

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

- ➢ **Nivel de Prueba**

Es la escala más 'micro' de las pruebas; para probar funciones especiales o módulos de código. Normalmente realizada por el programador y no por el probador, ya que requiere un conocimiento detallado del interior y el diseño de programas y del código. No es siempre fácil realizarse a menos que la aplicación disponga de un buen diseño de arquitectura con código muy estricto; podrá requerir el desarrollo de módulos piloto de pruebas. (30)

- ➢ **Técnica de Prueba**

Se aplicarán las **Pruebas de Funcionalidad** las cuales se centran en la validación de las funciones, métodos, servicios y caso de uso. Se realiza a cada funcionalidad terminada con el objetivo de verificar el cumplimiento y de esta forma satisfacer las peticiones.

- ➢ **Tipo de prueba**

Dentro de las pruebas de funcionalidad se realizan las **Pruebas Funcionales,** una prueba funcional es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático. (31)

- ➢ **Método de prueba**

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa.

Los casos de prueba de la caja negra pretenden demostrar que:

- ✓ Las funciones del software son operativas.
- ✓ La entrada se acepta de forma correcta.
- ✓ Se produce una salida correcta.
- ✓ La integridad de la información externa se mantiene. (32)

Para diseñar los casos de prueba de Caja Negra se utilizó la Técnica de la Partición de Equivalencia la cual presenta la partición equivalente como un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Típicamente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

El objetivo de partición equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software. Se toma un riesgo porque se escoge no probar todo. Así que se necesita tener mucho cuidado al escoger las clases. (33)

Método de prueba de Caja Blanca

Las pruebas de Caja Blanca, también conocidas como pruebas de caja de cristal o pruebas estructurales, se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Estas son consideradas como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado la disminución del número de errores existentes y aportando mayor calidad y confiabilidad al producto. Mediante su aplicación, el ingeniero de software puede obtener casos de prueba que garanticen que: (29)

- ✓ Se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- ✓ Se ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- ✓ Se ejecuten todos los bucles en sus límites operacionales.
- ✓ Se ejerciten las estructuras internas de datos para asegurar su validez.

Dentro de esta clasificación se encuentra la prueba del camino básico, que es una técnica de prueba que permite obtener una medida de la complejidad lógica de un diseño y usarla como guía para la definición de un conjunto básico. Esta consiste en derivar casos de prueba a partir de un conjunto de caminos independientes por los cuales puede circular el flujo de control. Para ello es necesario construir el grafo de flujo asociado y calcular su complejidad ciclomática. Una vez identificados los caminos independientes se procede a preparar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

La complejidad ciclomática del grafo puede ser calculada de las siguientes formas:

V (G) = Número de regiones del grafo de flujo.

V (G) = A – N + 2.

V (G) = P + 1.

Donde V (G) es la complejidad ciclomática de un grafo G, A es el número de aristas, N es el número de nodos contenidos en el grafo y P es el número de nodos predicados contenidos en el grafo.

Los nodos predicados son aquellos a partir de los cuales se puede tomar más de un camino. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. Como se mencionó anteriormente, entre los métodos de prueba de Caja Blanca se encuentra el del camino simple, el cual se aplica a fragmentos del código fuente de la aplicación. Por ser una de las más significativas de la extensión, se determinó aplicar este método a la funcionalidad que genera las matrices de trazabilidad. Para la función construirMatriz (), perteneciente a la clase GenerarMatriz.java, fueron identificados y enumerados los bloques de ejecución como se muestra a continuación.

```
private Object[][] ConstruirMatriz(int col) throws SQLException {
    LinkedList<String> filas = null;                                                    1
    if(soloProyecto.isSelected())                                                       2
    filas = requisitoActions.GetListadoNombreRequisito_DadoProyecto(proyectos.getSelectedItem().toString());   3
    else if(soloModulo.isSelected())                                                    4
    filas = requisitoActions.GetListadoNombreRequisito_DadoModulo
    (moduloActions.GetIdModuloDadoNombreModulo(modulos.getSelectedItem().toString()));   5
    Object[] columnas = null;                                                           6
    if (DimeTipo(col).equals("elemento")) {                                             7
        columnas = elementoActions.GetNombreElemento(GetNombreElementoDadoOpcion(Hasta.getSelectedIndex(), Hasta));}   8
    Object[][] res = new Object[filas.size() + 1][columnas.length + 1];                 9
    for (int i = 0; i < filas.size(); i++) {                                            10
        res[i + 1][0] = filas.get(i).toString();                                        11
        for (int j = 0; j < columnas.length; j++) {                                     12
            res[0][j + 1] = columnas[j].toString();                                     13
            if (requisitoActions.DimeSiExisteRelacion(columnas[j].toString(), filas.get(i).toString(),
            GetNombreElementoRelacionDadoOpcion(Hasta.getSelectedIndex(), Hasta))) {    14
                res[i + 1][j + 1] = "x";                                                15
            }}}
    matrizDiagrama = res;                                                               16
    return res;                                                                         17
    }
```

Ilustración 23 Método de Prueba Generar Matriz

De ese modo se obtuvieron 17 bloques, se dibujó el grafo de flujo asociado y se determinó el camino básico. Como se muestra en el grafo de flujo las aristas indican los posibles caminos a seguir a partir del nodo correspondiente. Partiendo del camino básico determinado, fue aplicado uno de los tres métodos para calcular la complejidad ciclomática, específicamente el método V (G) = A – N + 2, aunque todos ellos arrojan al mismo resultado. Se obtuvieron 22 aristas y 17 nodos, quedando la fórmula de la siguiente forma: V (G) = 22 – 17 + 2. Por lo tanto la complejidad ciclomática tiene un valor de 7, lo cual significa que existen 7 posibles casos de ejecución para la función.

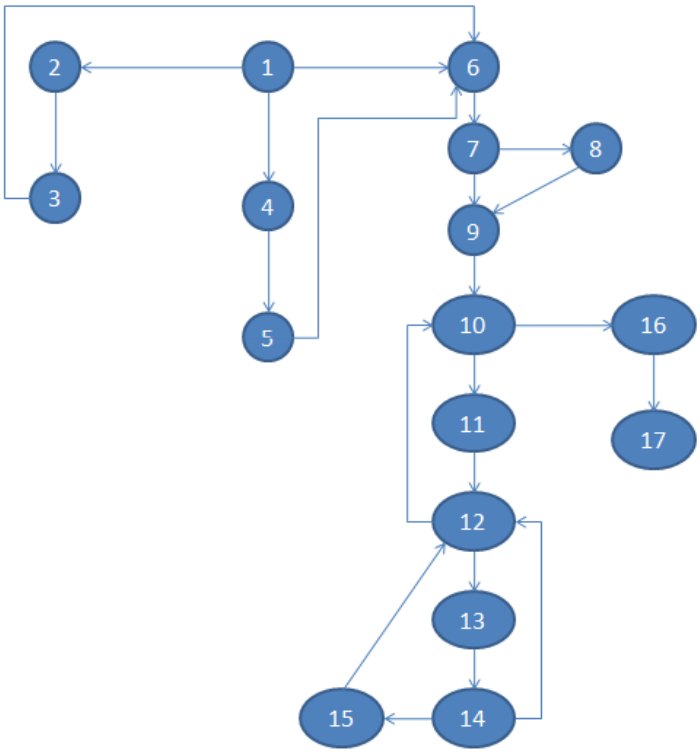


Ilustración 24 Grafo de flujo que muestra el camino básico de la función “Generar Matriz”.

Casos de Pruebas

Son pruebas que se llevan a cabo directamente en la interfaz del software. Se comprueba el correcto funcionamiento de los componentes del sistema de información, analizando las entradas y salidas y verificando que el resultado es el esperado. Se consideran exclusivamente las entradas y salidas del sistema sin preocuparse por la estructura interna del mismo. Errores que se pretenden detectar: funciones incorrectas o ausentes, errores de interfaz, errores en las estructuras de datos, errores de rendimiento, errores de inicialización y terminación. A continuación se presenta el caso de prueba, para el caso de uso Establecer Conexión con la Base de Datos. Estas pruebas se realizan a través de una matriz de datos, para comprobar que la extensión funcione de forma correcta, donde:

V: indica válido, I: indica inválido, NA: que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.

- ✓ En la siguiente tabla se muestran las variables V1, V2, Vn…, que representan valores de entrada de datos para los casos de pruebas.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
V1	*nombre del host*	*campo de texto*	*No*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*
V2	*puerto*	*campo de texto*	*No*	*Alfanumérico de 1 a 255 -.*
V3	*nombre recurso*	*campo de texto*	*No*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*
V4	*usuario*	*campo de texto*	*No*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*
V5	*contraseña*	*campo de texto password*	*No*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*

Tabla 8 Variables de entrada para los Casos de Prueba

Escenario	Descripción	V1	V2	V3	V4	V5	Respuesta del sistema	Flujo central
EC 1.1 Insertar datos de conexión. Correctamente	Se adicionan correctamente los datos.	*V (10.56.17.4)*	*V (5432)*	*V (datos)*	*V (postgres)*	*V (Postgres)*	Se realiza la conexión correctamente.	1- Se escoge la opción Administración de Requisitos.
								2- Se llenan los datos.

								3- Se escoge la opción siguiente
EC 1.2 Insertar datos de conexión. Con campos en blanco	Se adicionan los datos dejando campos en blancos. Se muestra un mensaje indicando el error.	*N/A*	*V (5432)*	*V (datos)*	*V (postgres)*	*V (Postgres)*	Se muestra un mensaje indicando que existen campos requeridos en blanco.	1- Se escoge la opción Administración de Requisitos. 2- Se llenan los datos. 3- Se escoge la opción siguiente
		V (10.56.17.4)	*N/A*	*V (datos)*	*V (postgres)*	*V (Postgres)*		
		V (10.56.17.4)	*V (5432)*	*N/A*	*V (postgres)*	*V (Postgres)*		
		V (10.56.17.4)	*V (5432)*	*V (datos)*	*N/A*	*V (Postgres)*		
		V (10.56.17.4)	*V (5432)*	*V (datos)*	*V (postgres)*	*N/A*		
EC 1.3 Insertar datos de conexión. Con datos no válidos	Se adicionan los datos dejando campos inválidos. Se muestra un mensaje indicando el error.	*I (host no existente)*	*V (5432)*	*V (datos)*	*V (postgres)*	*V (Postgres)*	Se muestra un mensaje indicando que los datos son incorrectos.	1- Se escoge la opción Administración de Requisitos. 2- Se llenan los datos. 3- Se escoge la opción siguiente
		V (10.56.17.4)	*I (puerto no existente)*	*V (datos)*	*V (postgres)*	*V (Postgres)*		
		V (10.56.17.4)	*V (5432)*	*I (recurso no existente)*	*V (postgres)*	*V (Postgres)*		
		V (10.56.17.4)	*V (5432)*	*V (datos)*	*I (usuario incorrecto)*	*V (Postgres)*		
		V (10.56.17.4)	*V (5432)*	*V (datos)*	*V (postgres)*	*I (contraseña incorrecta)*		
EC 1.4 Cancelar. Insertar	Se cancela la operación Insertar datos	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	Se cancela la operación insertar datos	1- Se escoge la opción

datos de conexión.	de la conexión.						para la conexión.	Administración de Requisitos.
								2- Se escoge la opción siguiente

Tabla 9 Caso de Prueba - CU Establecer Conexión con la Base de Datos

Después de realizar las pruebas de Caja Negra al Caso de Prueba asociado a establecer la conexión con la base de datos, se comprobó el correcto funcionamiento de la extensión y la correcta validación de los campos, verificando que solo se acepten los caracteres válidos para los mismos. Cada dificultad detectada en el desarrollo de la extensión y resueltas a raíz del trabajo de los desarrolladores, fueron recogidas en la planilla de No Conformidades. En la siguiente tabla se muestra un resumen de las dificultades encontradas:

PD: Pendiente **RA:** Resuelta

Fecha	Versión	Caso de Prueba	Cant. de No Conformidad	Cant. de No Conformidad PD	Cant. De No Conformidad RA
15/01/2012	1.0	CU-Establecer conexión con la BD	4	0	4
3/02/2012	1.1	CU- Establecer conexión con la BD	0	0	0

Tabla 10 Plantilla de No Conformidades

3.4 Conclusiones parciales del capítulo

En este capítulo se realizó el modelo de implementación con el propósito de mostrar los componentes del sistema y sus relaciones, a través del diagrama de componentes. Además, se obtuvo el diagrama de despliegue de la extensión, también se diseñó el diagrama de componente en el cual se evidencia la estructura de la extensión y, por último se realizaron pruebas para validar que los requisitos fueron implementados correctamente a través del método de caja negra, aplicando la técnica de partición de equivalencia y se validó el correcto funcionamiento del código empleado mediante el método de caja blanca y la técnica de camino básico.

CONCLUSIONES

Para dar cumplimiento a los objetivos planteados durante el presente trabajo de diploma se obtuvieron los siguientes resultados generales.

- ✓ Se analizaron los aspectos teóricos fundamentales sobre la extensión de la herramienta *"Visual Paradigm for UML"* para la administración de requisitos.
- ✓ Se definió la plataforma tecnológica para el desarrollo de la extensión, con el objetivo de lograr una mayor eficiencia en el proceso de desarrollo.
- ✓ Se realizó el análisis diseño e implementación de la extensión.
- ✓ Se probó el correcto funcionamiento de la extensión, a través de las pruebas de caja negra realizadas mediante los casos de prueba.

RECOMENDACIONES

Para esta primera versión de la extensión de la herramienta "*Visual Paradigm for UML*" para la administración de requisitos, se cumplieron los objetivos planteados durante todo el desarrollo. Para próximas versiones se recomienda:

- ✓ Migrar todo el código de acceso a datos para un ORM (Mapeo Relacional de Objetos) soportado por el lenguaje de programación, por lo que se propone JPA (API de Persistencia de Java).

REFERENCIAS

1. *Centro Tecnologías de Gestión de Datos* http://gespro.datec.prod.uci.cu/

2. *Calisof t*http://calisoft.uci.cu/attachments/046_0523_Pol%C3%ADticas.pdf

3. *rodolfoquispe* http://www.rodolfoquispe.org/blog/que-es-la-ingenieria-de-requisitos.php

4. *Eva.uci.cu*
http://eva.uci.cu/file.php/161/Documentos/Materiales_basicos/Materiales_basicos_de_la_Unidad_2/Sommerville_Parte_II_Requerimientos.pdf

5. *Sparxsystems* http://www.sparxsystems.com.ar/

6. *www.scribd.com* http://www.scribd.com/doc/31440864/Metodologia-RUP

7. *Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa usando OpenUP.* 2008

8. *Ciberaula* http://java.ciberaula.com/articulo/tecnologia_java/

9. *Comunidad Netbeans* http://netbeans.org/community/releases/68/

10. *Aprenda JAVA como si estuviera en primero.*2000

11. *Wikipedia* http://es.wikipedia.org/wiki/Herramienta_CASE

12. *"Visual Paradigm for UML"* http://www.visual-paradigm.com

13. *Biblioteca.uci.cu*
http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04358_11/1/TD_04358_11.pdf

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

15. *The Unified Modeling Language User Guide. Massachusetts : Addison-Wesley Longman Inc.*19980-201-57168-4

16. *elvex.ugr.es* http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf

17. *www.sparxsystems.com.ar* http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html

18. *wikipedia.org* http://es.wikipedia.org/wiki/Caso_de_uso

19. *Use Cases Patterns and Blueprints. s.l. : Addison Wesley Professional.*20040-13-145134-0

20. *www.planetacodigo.com* http://www.planetacodigo.com/wiki/glosario:matriz_de_trazabilidad

21. *wikipedia.org* http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

22. *msdn.microsoft.com* http://msdn.microsoft.com/es-es/library/bb972272.aspx

23. *ldc.usb.ve* http://ldc.usb.ve/~teruel/ci3711/patron3a/index.html

24. *www.buenastareas.com* http://www.buenastareas.com/ensayos/Diagrama-Secuencia/4467844.html

25. *wikipedia.org* http://es.wikipedia.org/wiki/Modelo_relacional

26. *wikipedia.org* http://es.wikipedia.org/wiki/Diagrama_de_componentes

27. *wikipedia.org* http://es.wikipedia.org/wiki/C%C3%B3digo_fuente

28. *serk.kualtus.com* http://serk.kualtus.com/codigo.htm

29. *ingpau.blogspot.com* http://ingpau.blogspot.com/2007/09/etapa-de-pruebas.html

30. *www.itbuilder.com.mx* http://www.itbuilder.com.mx/blogs/edgar.alvarado/post/Niveles-de-Prueba-(Levels-of-Testing).aspx

31. *es.wikipedia.org* http://es.wikipedia.org/wiki/Pruebas_funcionales

32. *www.buenastareas.com* http://www.buenastareas.com/ensayos/Pruebas-De-Caja-Negra/2060455.html

33. *gemini.udistrital.edu.co*
http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html

BIBLIOGRAFÍAS

1. *Centro Tecnologías de Gestión de Datos* http://gespro.datec.prod.uci.cu/

2. *Calisof t*http://calisoft.uci.cu/attachments/046_0523_Pol%C3%ADticas.pdf

3. *rodolfoquispe* http://www.rodolfoquispe.org/blog/que-es-la-ingenieria-de-requisitos.php

4. *Eva.uci.cu*
http://eva.uci.cu/file.php/161/Documentos/Materiales_basicos/Materiales_basicos_de_la_Unidad_2/Sommerville_Parte_II_Requerimientos.pdf

5. *Sparxsystems* http://www.sparxsystems.com.ar/

6. *www.scribd.com* http://www.scribd.com/doc/31440864/Metodologia-RUP

7. *Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa usando OpenUP.* 2008

8. *Ciberaula* http://java.ciberaula.com/articulo/tecnologia_java/

9. *Comunidad Netbeans* http://netbeans.org/community/releases/68/

10. *Aprenda JAVA como si estuviera en primero.*2000

11. *Wikipedia* http://es.wikipedia.org/wiki/Herramienta_CASE

12. *"Visual Paradigm for UML"* http://www.visual-paradigm.com

13. *Biblioteca.uci.cu*
http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_04358_11/1/TD_04358_11.pdf

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

15. *The Unified Modeling Language User Guide. Massachusetts : Addison-Wesley Longman Inc.*19980-201-57168-4

16. *elvex.ugr.es* http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf

17. *www.sparxsystems.com.ar* http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html

18. *wikipedia.org* http://es.wikipedia.org/wiki/Caso_de_uso

19. *Use Cases Patterns and Blueprints. s.l. : Addison Wesley Professional.*20040-13-145134-0

20. *www.planetacodigo.com* http://www.planetacodigo.com/wiki/glosario:matriz_de_trazabilidad

21. *wikipedia.org* http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

22. *msdn.microsoft.com* http://msdn.microsoft.com/es-es/library/bb972272.aspx

23. *ldc.usb.ve* http://ldc.usb.ve/~teruel/ci3711/patron3a/index.html

24. *www.buenastareas.com* http://www.buenastareas.com/ensayos/Diagrama-Secuencia/4467844.html

25. *wikipedia.org* http://es.wikipedia.org/wiki/Modelo_relacional

26. *wikipedia.org* http://es.wikipedia.org/wiki/Diagrama_de_componentes

27. *wikipedia.org* http://es.wikipedia.org/wiki/C%C3%B3digo_fuente

28. *serk.kualtus.com* http://serk.kualtus.com/codigo.htm

29. *ingpau.blogspot.com* http://ingpau.blogspot.com/2007/09/etapa-de-pruebas.html

30. *www.itbuilder.com.mx* http://www.itbuilder.com.mx/blogs/edgar.alvarado/post/Niveles-de-Prueba-(Levels-of-Testing).aspx

31. *es.wikipedia.org* http://es.wikipedia.org/wiki/Pruebas_funcionales

32. *www.buenastareas.com* http://www.buenastareas.com/ensayos/Pruebas-De-Caja-Negra/2060455.html

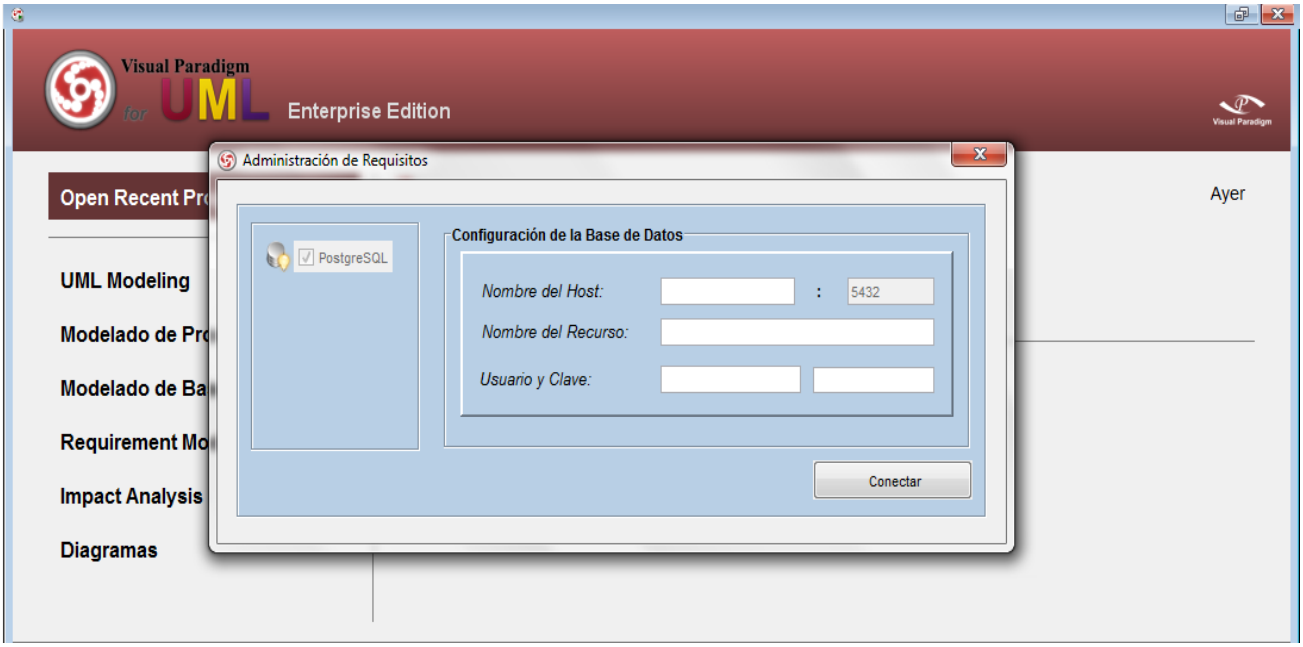
33. *gemini.udistrital.edu.co*
http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html

34. *calisoft.uci.cu* http://calisoft.uci.cu/index.php/proceso-de-mejora/46

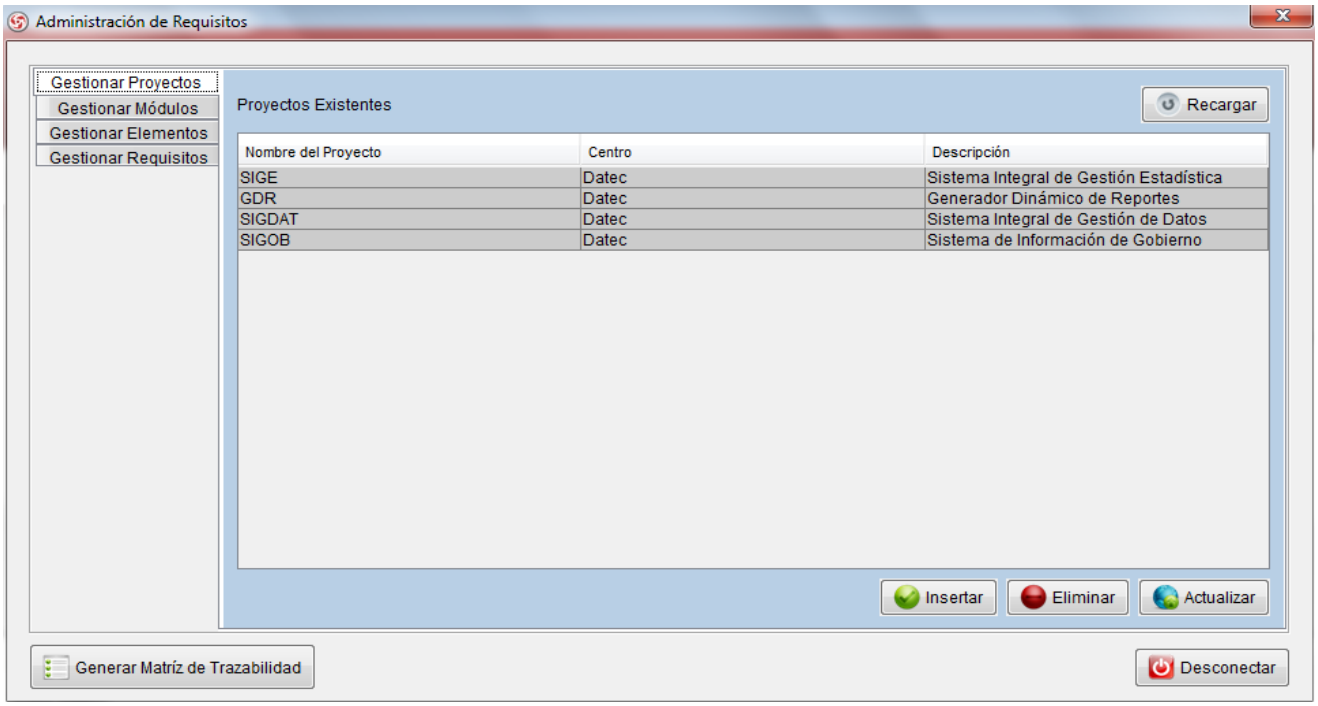
ANEXOS

Imágenes de la extensión de la herramienta "*Visual Paradigm for UML*" para la administración de requisitos.

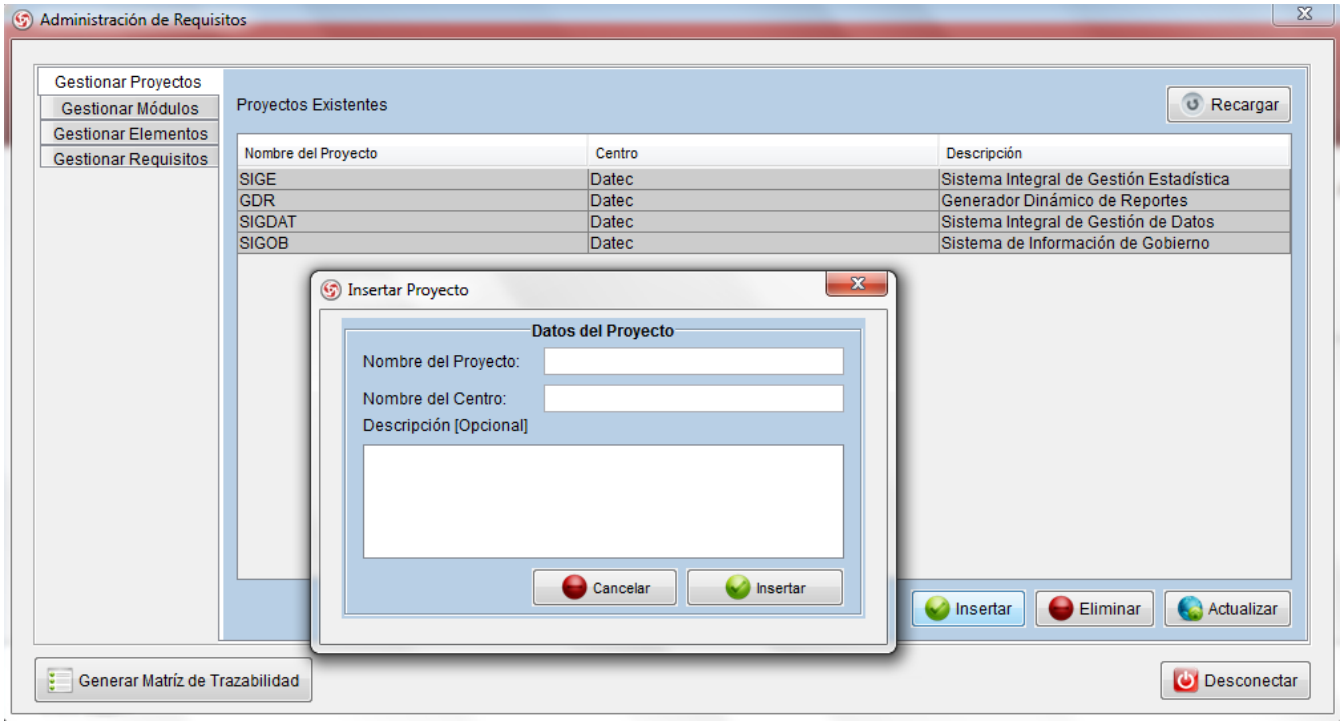
Interfaz de usuario para la establecer la conexión con la Base de Datos de la extensión:



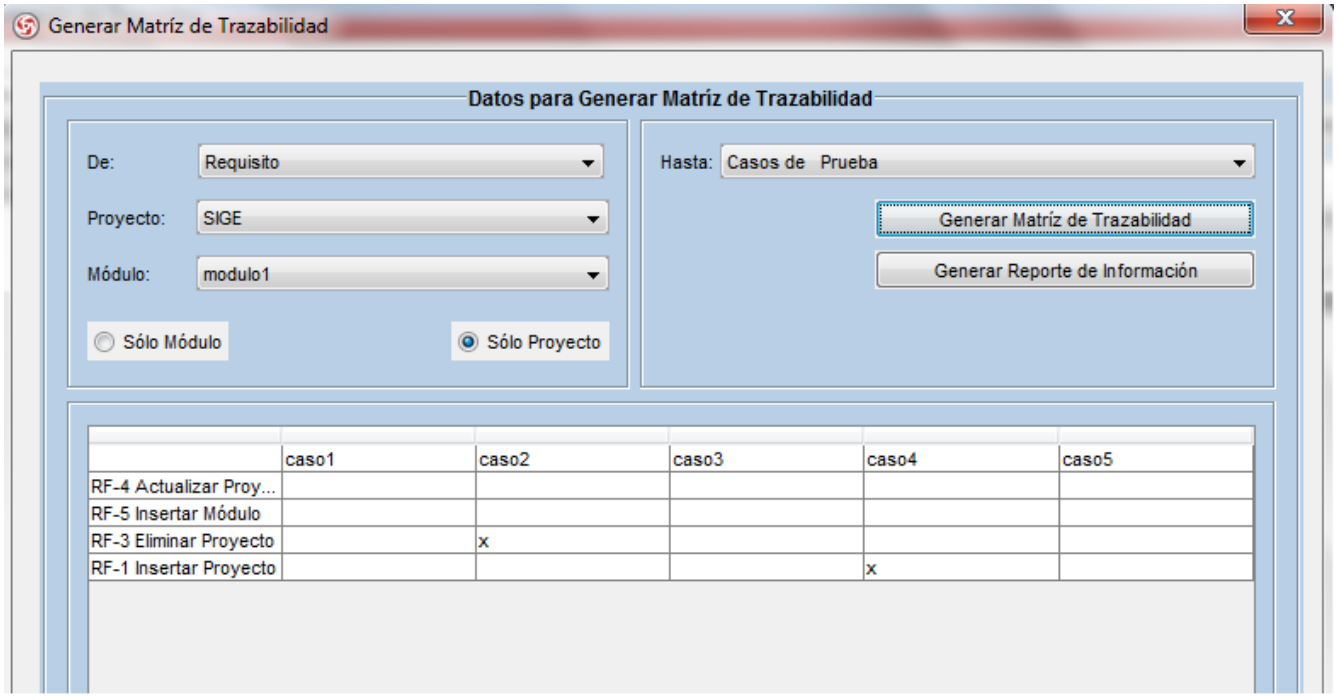
Interfaz de usuario Principal de donde parten todas las interfaces:



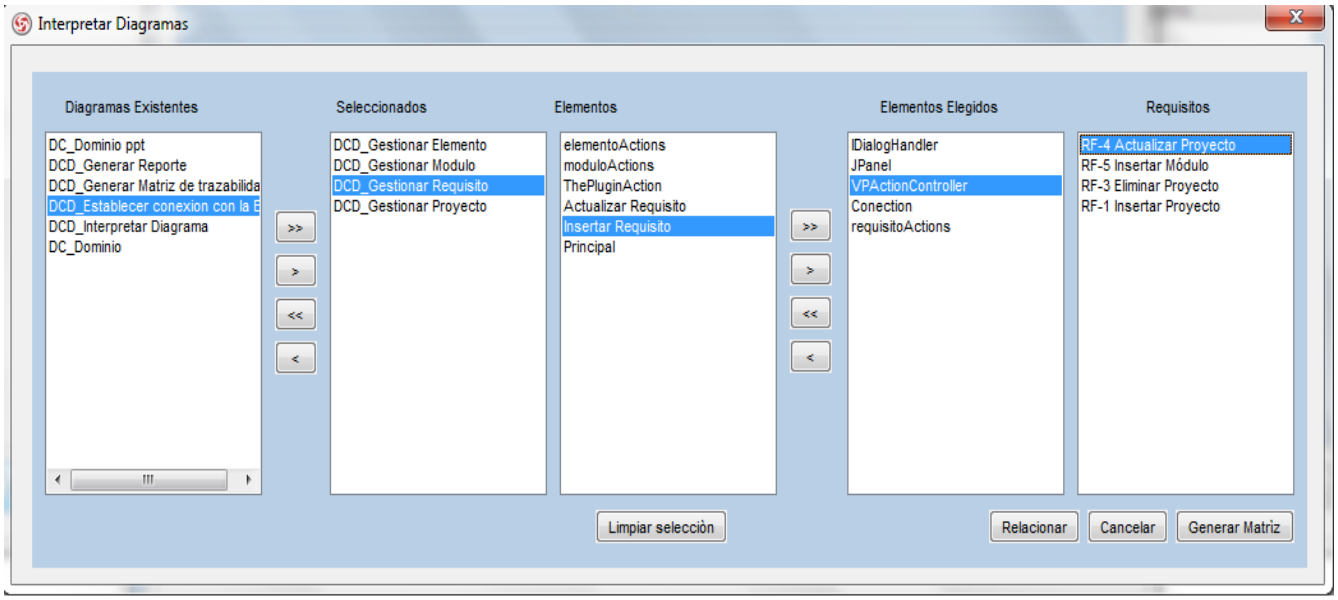
Interfaz de usuario para insertar un nuevo Proyecto:



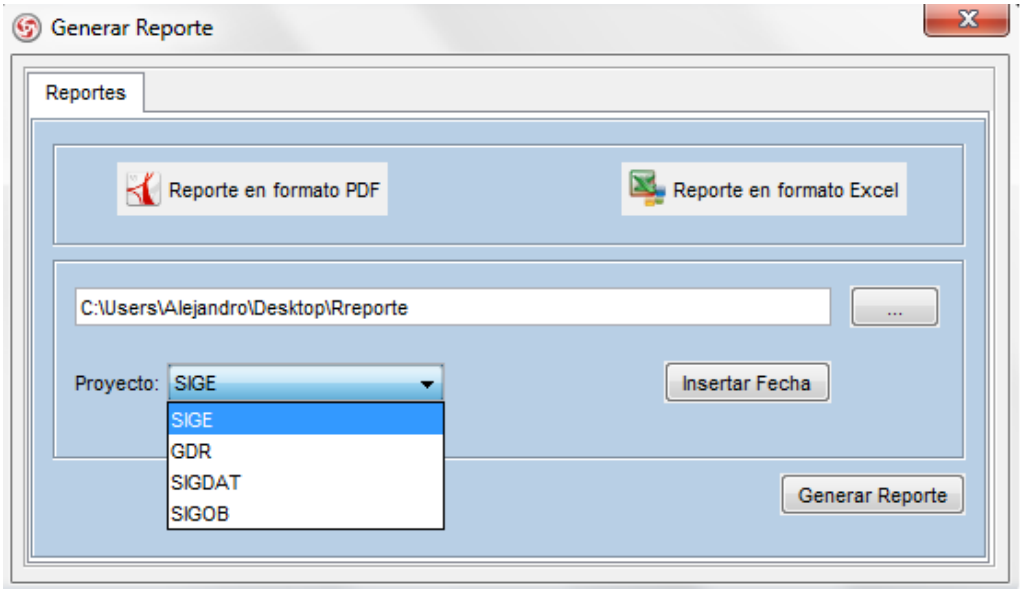
Interfaz de usuario que permite generar matrices de trazabilidad:



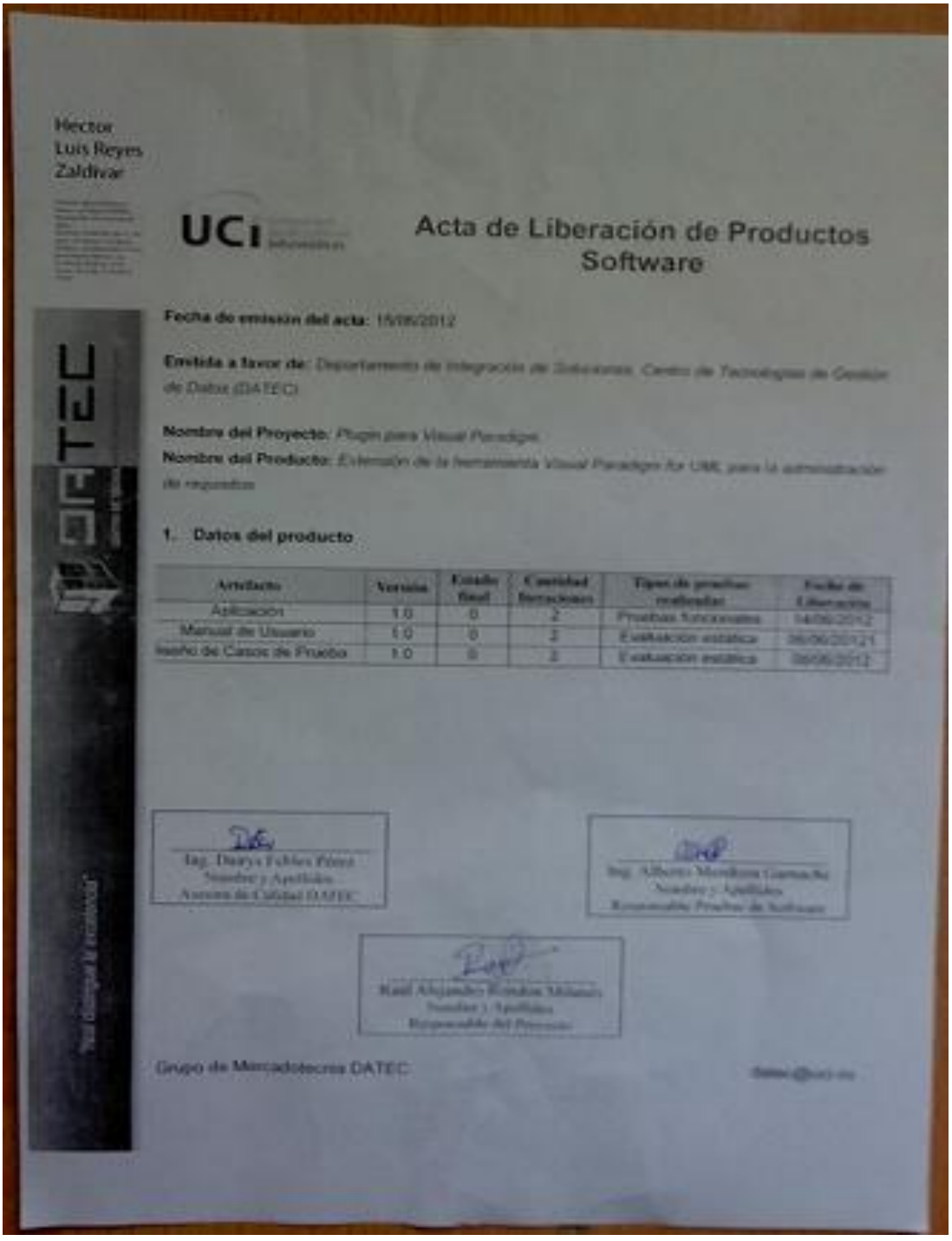
Interfaz de usuario que permite establecer la relación de requisitos con diagramas de "*Visual Paradigm for UML*":



Interfaz de usuario que permite generar reportes de información incluyendo la matriz de trazabilidad para el cliente:



Acta de liberación de la extensión.



Hector
Luis Reyes
Zaldivar

UCI

Acta de Liberación de Productos Software

Fecha de emisión del acta: 15/06/2012

Emitida a favor de: Departamento de Integración de Soluciones. Centro de Tecnologías de Gestión de Datos (DATEC)

Nombre del Proyecto: *Plugin para Visual Paradigm*

Nombre del Producto: *Extensión de la herramienta Visual Paradigm for UML para la administración de requisitos*

1. Datos del producto

Artefacto	Versión	Estado final	Cantidad Iteraciones	Tipos de pruebas realizadas	Fecha de Liberación
Aplicación	1.0	0	2	Pruebas funcionales	14/06/2012
Manual de Usuario	1.0	0	2	Evaluación estática	06/06/2012
Diseño de Casos de Prueba	1.0	0	2	Evaluación estática	06/06/2012

Ing. Daarys Fables Pérez
Nombre y Apellidos
Asesora de Calidad DATEC

Ing. Alberto Montilla Camacho
Nombre y Apellidos
Responsable Pruebas de Software

Raúl Alejandro Rondón Molano
Nombre y Apellidos
Responsable del Proyecto

Grupo de Mercadotecnia DATEC

GLOSARIO DE TÉRMINOS

Administración de requisitos: la administración de todos los requisitos recibidos o generados por el proyecto, incluyendo requisitos técnicos y no técnicos, así como esos requisitos levantados en el proyecto por la organización.

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

Proveedor de requisitos: persona que provee de requisitos a algún miembro del proyecto (para los requisitos del cliente el proveedor es el cliente, para los requisitos técnicos es algún miembro del proyecto autorizado para ello).

Requisito: una condición o capacidad necesitada por el usuario para resolver un problema o lograr un objetivo. Una condición o capacidad que debe poseer un producto o componente de producto para satisfacer un contrato, estándar, especificación, u otros documentos obligatorios formales. Una representación documentada de una condición o capacidad.

Requisitos técnicos: propiedades (atributos) de productos o servicios a ser adquiridos o desarrollados.

Trazabilidad: asociación distinguible entre dos o más entidades lógicas como requisitos, elementos del sistema, verificaciones, o tareas.

Trazabilidad bidireccional: asociación entre dos o más entidades lógica que es observable en cualquier dirección (desde y hacia una entidad).

Trazabilidad de requisitos: una asociación distinguible entre requisitos y requisitos relacionados, implementación, y verificación. (Ver también, trazabilidad bidireccional y traceabilidad).

Extensión de la herramienta o Plugin: módulo que puede ser de Hardware o de Software que añade características o funcionalidades a una aplicación más grande.