Universidad de las Ciencias Informáticas

FACULTAD 6

Título: "Extensión de la herramienta CASE *Visual Paradigm for UML* para la Administración de Requisitos versión 2.0."

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

Autor:

Jorge Ernesto Linares Triana.

Tutores:

Ing. Viviana Cruz Amaro.

Ing. Alberto Mendoza Garnache.

La Habana, junio de 2013

"Año 55 de la Revolución"

“Al mundo no le importará tu autoestima. El mundo esperará que logres algo, independientemente de que te sientas bien o no contigo mismo”

Bill Gates

Declaración de Autoría

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

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

Jorge Ernesto Linares Triana — Autor

Ing. Viviana Cruz Amaro — Tutora

Ing. Alberto Mendoza Garnache — Tutor

Datos de contacto

Autor: Jorge Ernesto Linares Triana
Universidad de las Ciencias Informáticas
La Habana, Cuba
E-mail: jelinares@estudiantes.uci.cu

Tutora: Ing. Viviana Cruz Amaro
Universidad de las Ciencias Informáticas
La Habana, Cuba
E-mail: vamaro@uci.cu

Tutor: Ing. Alberto Mendoza Garnache
Universidad de las Ciencias Informáticas
La Habana, Cuba
E-mail: agarnache@uci.cu

Agradecimientos

Primeramente quiero agradecer a mi mamá Marlene y a mi papá Martín, por su amor y cariño, por su apoyo en todas mis decisiones, por el esfuerzo y trabajo con el cual me han criado haciendo de mí lo que ahora soy.

A mi hermano Ernesto, a mi abuela Daysi, a mi primo Nano, a Yendry, a mis tíos, por su infinito amor y por existir.

A todos mis compañeros que estuvieron conmigo en estos cinco años de carrera, en los buenos y en los malos momentos, en especial quiero mencionar a: Marvel, Osvaldo, Ernesto, Liandry, Bazán, Alejandro, Sandy, Julio, Rolando, René, Osman, Gleivis y Sureny, en general a toda la gente de los grupos cinco y seis que compartieron momentos conmigo durante este largo proceso.

A mi tutor Garnache por estar siempre pendiente de todos los detalles, guiándome en todo el desarrollo del trabajo. A Vivi por ser más que mi tutora, ser mi amiga, por regañarme en cada momento que lo necesitaba, por enseñarme a ser una mejor persona y por guiarme hasta aquí.

Dedicatoria

A mis padres Marlene y Martín por su apoyo incondicional.

A mi abuela por su gran amor y paciencia.

A mi hermano por estar siempre conmigo y espero ser un buen ejemplo para él.

A mi primo por ser como mi hermano.

A mi prima por estar siempre pendiente de mí.

A toda mi familia por esperar siempre lo mejor de mí.

A todos los que de una forma u otra me ayudaron a crecer en estos cinco años de universidad.

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, específicamente en el Departamento de Integración de Soluciones, el Grupo de Análisis desarrolla los productos utilizando la herramienta CASE "*Visual Paradigm for UML*" como apoyo para la realización del análisis del desarrollo de software. Dicho grupo disponía para la Administración de Requisitos de una extensión de esta herramienta en su versión 1.0, la cual presentaba algunas limitaciones. El presente trabajo tuvo como objetivo realizar la versión 2.0 de la extensión, la cual permitió hacer mejoras en aspectos como la usabilidad, la generación de las matrices de trazabilidad y en la información emitida en la generación de reportes. Se incorporaron funcionalidades relacionadas con los subprocesos: administración de requisitos y traceo. Con esta extensión se garantiza la correcta administración de los requisitos en el proceso de desarrollo de software. Como resultado se obtuvo un prototipo funcional de la extensión de la herramienta CASE "*Visual Paradigm for UML*" versión 2.0 para la Administración de Requisitos. La extensión implementada se sometió a pruebas 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: Administración de Requisitos, CASE, extensión, matrices de trazabilidad, "*Visual Paradigm for UML".*

Tabla de Contenidos

INTRODUCCIÓN

El surgimiento de los sistemas de cómputo trajo consigo el origen de componentes que hicieron posible la realización de tareas específicas, estos se encargaban de manejar las partes físicas que comprendían dichos sistemas. El control de las operaciones estaba parcialmente integrado en el equipo, dicho control era realizado por un circuito que requería un cableado específico para cada aplicación. Hasta ese momento, no se percibía una diferencia sustancial entre el equipo y el control de las operaciones.

En sus inicios el software no se consideraba un producto sino un añadido, se diseñaba a la medida para cada aplicación, pues tenía instrucciones muy específicas y sencillas, además de ser producido por la misma organización o persona que lo utilizaba. Con el tiempo los sistemas de cómputo se fueron haciendo más complejos introduciendo conceptos como la multiprogramación y multiusuario surgiendo una nueva relación hombre-máquina. Estas nuevas transformaciones trajeron consigo todo un mundo de nuevas aplicaciones de software las cuales podían captar, analizar y proveer un resultado eficiente en el menor tiempo posible; pero existía un problema fundamental, la puesta en funcionamiento de estos sistemas acarreaban errores de programación y desorganización del código fuente. Debido a que el desarrollo de estas sentencias se realizaba sin previa planificación, tenían que ser corregidas cuando se detectaban fallos, ser modificadas cuando cambiaban los requisitos o adaptadas a nuevos dispositivos hardware que se desarrollaran. Como solución a estos problemas surge la disciplina Ingeniería del Software que tiene como objetivo aplicar un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, para la obtención de resultados óptimos de la forma más eficiente.

Cuba no ha quedado exenta del desarrollo de software un ejemplo de esto es el surgimiento de la Universidad de las Ciencias Informáticas (UCI) que ha tenido como objetivo principal llevar adelante la informatización del país en función de los objetivos propuestos y dirigido a cumplir con el desarrollo científico y económico; teniendo en cuenta los intereses del país, el potencial tecnológico y humano disponibles. Para llevar a cabo este proceso, la UCI ha creado una infraestructura en la cual se encuentran los centros de desarrollo, uno de estos es el Centro de Tecnologías de Gestión de Datos (DATEC) que tiene como objetivo la creación de bienes y servicios informáticos relacionados con la gestión de datos, teniendo como propósito fundamental apoyar el proceso de toma de decisiones. Además el centro es especializado en tecnologías de Gestión de Datos y Bioinformática teniendo un gran prestigio en el ámbito nacional.

Introducción

El centro está dividido en diferentes departamentos, uno de ellos es el Departamento de Integración de Soluciones, que tiene como objetivo la creación de productos y servicios que incluyen productos para la gestión de proyectos de software, además servicios de integración de soluciones y desarrollo de las herramientas necesarias para crear soluciones integrales de análisis de datos. La estructura organizativa del departamento está dividida en tres grupos: Análisis, Arquitectura y Desarrollo. El Grupo de Análisis es el encargado de realizar actividades en el tiempo establecido y con la calidad requerida.

La Dirección de Calidad de Software de la universidad establece políticas organizacionales que guían el proceso de desarrollo de software con el objetivo de obtener productos de software con la calidad requerida, cumpliendo con el nivel dos de CMMI, el cual comprende siete áreas de procesos: 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 mantener bajo control los requisitos que el producto a desarrollar deberá satisfacer. Las prácticas incluidas apuntan a que los requisitos no solo estén claramente identificados, sino también que todos los involucrados en el proyecto estén de acuerdo en su significado. Adicionalmente, los requisitos deben ser la entrada a las actividades de planificación. Otro aspecto importante incluido en el área de Administración de Requisitos es el de trazabilidad bidireccional, cuando los requisitos son correctamente administrados deberíamos estar en condiciones de relacionar cuál ha sido el origen de los requisitos, cuál es la relación entre los requisitos de bajo nivel y los de alto nivel, cuáles son los artefactos relacionados con los requisitos y cuáles componentes del producto implementan cada requisito. Esta práctica es sumamente importante para poder realizar un buen análisis de impacto ante posibles cambios, y fundamental para poder determinar si el alcance del proyecto ha sido cubierto o no.

En el Departamento de Integración de Soluciones el Grupo de Análisis lleva a cabo un conjunto de actividades como parte del proceso de desarrollo de software. Dicho grupo de trabajo dispone para la Administración de Requisitos de la extensión de la herramienta CASE “*Visual Paradigm for UML*” (VP) en su versión 1.0, la cual permite una vez establecida la conexión a la base de datos, la gestión de proyectos, módulos, requisitos y elementos de la base de datos de cada proyecto. Además brinda la posibilidad de generar matrices de trazabilidad, interpretar diagramas modelados en la herramienta y generar reportes.

Introducción

La extensión en su primera versión garantiza parcialmente la trazabilidad bidireccional entre los requisitos, sus productos de trabajo derivados y los planes del proyecto, ya que solo soporta algunos de los elementos predefinidos a tracear dentro del proyecto, especificados en la guía de trazabilidad del proceso de mejora; lo cual puede incidir negativamente en la toma de decisiones sobre los cambios solicitados por el cliente o el equipo de desarrollo, a la hora de determinar qué se necesita cambiar, cuántos elementos se deben modificar, dónde se ubican dichos elementos, así como el esfuerzo y tiempo de desarrollo que implican dichos cambios.

El proceso de generación de la matriz de trazabilidad en la herramienta no se realiza de manera objetiva, ya que las relaciones entre los requisitos y los elementos de los diagramas se establecen durante este proceso, lo que provoca que el usuario se adelantaría a los resultados de dicho proceso. Por otro lado no se almacenan en la base de datos las dependencias que se establecen entre los requisitos y los elementos de los diagramas, provocando que se tenga que hacer de forma reiterativa. Otras de las deficiencias que presenta la extensión es que solo soporta el gestor de base de datos PostgreSQL y la información requerida para los usuarios, como las matrices de trazabilidad, no se muestran en los reportes generados por la extensión. Se puede afirmar que la extensión tiene algunas limitantes relacionadas con la seguridad, pues no existe una gestión de roles y usuarios para restringir el acceso a los módulos de los proyectos.

Por lo anteriormente descrito se plantea como **problema de la investigación:** ¿Cómo contribuir con el proceso de Administración de Requisitos en el desarrollo de productos de software?

Se define 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 la versión 2.0 de la extensión de la herramienta CASE "*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* versión 2.0".
- ✓ Realizar el diseño de la segunda versión 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.
- ✓ Aplicación de la primera versión de la extensión en proyectos reales del departamento.
- ✓ Elaboración de los artefactos según la metodología de desarrollo de software y requisitos mínimos definidos en los lineamientos para las tesis de la facultad.
- ✓ Identificación de los requisitos funcionales y no funcionales para el correcto funcionamiento de la extensión de la herramienta CASE "*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 prueba definidos a partir de los requisitos funcionales.
- ✓ Realización de pruebas funcionales y otros tipos de pruebas para comprobar el correcto funcionamiento de la extensión.

El presente documento se divide en tres capítulos:

Capítulo 1. Fundamento teórico de la extensión: En el capítulo se realiza un estudio relacionado a la Ingeniería de Requisitos, sus principales procesos y un análisis profundo del proceso Administración de Requisitos, además se estudian las principales herramientas existentes para la administración de requisitos. También se definen las principales herramientas y tecnologías necesarias para desarrollar la segunda versión de la extensión.

Capítulo 2. Análisis y diseño de la extensión: En el capítulo se define todo lo referente a las funcionalidades que debe garantizar la extensión en su segunda versión. Se construyen los artefactos correspondientes al análisis y diseño, modelando la estructura de la extensión.

Capítulo 3. Implementación y prueba de la extensión: En el capítulo se presentan 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 segunda versión de la extensión implementada.

CAPÍTULO 1: FUNDAMENTO TEÓRICO DE LA EXTENSIÓN

En este capítulo se realiza un estudio de la Ingeniería de Requisitos como disciplina de la Ingeniería del Software y los procesos que comprende, enfocándose principalmente en el proceso de Administración de Requisitos. Se realiza un análisis de los subprocesos que comprende la Administración de Requisitos, destacando al subproceso del mismo nombre, siendo el área donde se desarrolla la extensión. Se analizan las principales herramientas para la Administración de Requisitos realizando una comparación entre las mismas. Por último se abarca un estudio de la metodología, herramientas y tecnologías a utilizar para el desarrollo de la extensión.

1.1 Ingeniería de Requisitos de Software

Según el profesor Ian Sommerville[1] *"Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de este".* Puede definirse como una cualidad o característica que un determinado sistema o parte del mismo debe cumplir para satisfacer las necesidades de los clientes o de los usuarios finales.

Los requisitos deben especificarse por escrito como todo contrato o acuerdo entre dos partes antes de intentar comenzar la construcción del producto, sin ellos no podrá llevarse a cabo las etapas de diseño y construcción correctamente, además debe ser posible probarlos o verificarlos para comprobar si se cumplió con ellos o no; consistentes, que no entren en contradicción con otros requisitos; deben ser fáciles de leer y entender, o sea, concisos. Un requisito debe proporcionar la información suficiente para su comprensión y no debe ser ambiguo para no causarles confusión a los lectores.

Según la IEEE[2] los requisitos tienen dos niveles: Requisitos del Sistema y Requisitos de Software. Los Requisitos del Sistema son los que definen las funcionalidades y características que debe tener el sistema para satisfacer tanto los requisitos de negocio como los de usuario. Este tipo de requisitos van a servir como base para llevar a cabo la arquitectura, diseño y planes de pruebas del sistema. (1) Los Requisitos de Software se dividen en dos grupos: requisitos funcionales y no funcionales. Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Los requisitos no funcionales tienen que

[1] Ian Sommerville: Es un profesor de Ingeniería de Software de la Universidad de St. Andrews en Escocia.
[2] IEEE: Institute of Electrical and Electronics Engineers. Es una asociación técnico-profesional mundial dedicada a la estandarización.

ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares. (2) Según Ian Sommerville: *"La Ingeniería de Requisitos es el proceso de desarrollar una especificación de software. Las especificaciones pretenden comunicar las necesidades del sistema del cliente a los desarrolladores del sistema".* Existen varios criterios sobre los procesos que comprende la Ingeniería de Requisitos, uno de ellos es el de Ian Sommerville y otro el de Roger Pressman[3], la UCI adopta una combinación entre ambos criterios proponiendo los siguientes procesos: Estudio de viabilidad, Obtención, Análisis, Especificación, Validación y Administración de Requisitos.

Estudio de Viabilidad: Es un estudio rápido y orientado a conocer. Permite tener en cuenta si el sistema contribuye a los objetivos de la organización, si se puede implementar con la tecnología actual dentro de costo y tiempo, y si puede integrarse con otros sistemas existentes en la organización.

Obtención: Es el proceso mediante el cual los usuarios descubren, revelan, articulan y comprenden los requisitos que desean. En esta etapa los ingenieros de software trabajan con los clientes y usuarios para descubrir el dominio de la aplicación, los servicios que se deben proporcionar y las restricciones.

Análisis: La determinación de las características funcionales y de desempeño específicas del producto. El análisis puede estar basado en:

- ✓ El análisis de las necesidades del cliente, expectativas y limitaciones.
- ✓ Concepto de Operaciones.
- ✓ Ambientes proyectados de utilización (personas, productos, y procesos).
- ✓ Medidas de efectividad.
- ✓ La posibilidad de construir el software con el hardware y software existentes.
- ✓ La estimación del costo contra los beneficios que el software brindará a la organización.
- ✓ Permite tomar la decisión de si es factible realizar el sistema propuesto o no.
- ✓ Se realiza después de que se establecieron las necesidades del sistema.

Especificación: Es el proceso en el cual se realiza una descripción completa de las funcionalidades del sistema que será desarrollado, definiendo los requisitos funcionales y no funcionales, diagramas, modelos de sistemas y cualquier otra información que sirva de soporte para fases posteriores.

[3] Roger Pressman: Es un ingeniero de software estadounidense, autor y consultor.

Validación: Es similar al análisis, pues implica la confirmación por parte de los usuarios, de que los requisitos especificados cumplan con las características definidas. Es importante, pues los errores en los requisitos pueden conducir a costos excesivos si se descubren durante el desarrollo o después de la implantación.

Administración: Es el proceso de comprender y controlar los cambios en los requisitos. Además es el proceso que establece y mantiene un consenso entre el cliente y el grupo del proyecto en el cambio de los requisitos del sistema.

1.2 Administración de Requisitos

La Administración de Requisitos es vital en el desarrollo de un producto de software, pues es uno de los procesos de la Ingeniería de Requisitos más importantes. Este proceso gestiona todos los requisitos recibidos o generados por el proyecto, incluyendo tanto los requisitos funcionales como los no funcionales. (3) En el ciclo de vida de un proyecto dichos requisitos pueden ser modificados por lo que se establece el concepto de Administración de Requisitos, que es el tratamiento y control de las actualizaciones y cambios a los mismos. Debido a que un proyecto informático siempre se encuentra propenso a cambios, es necesario llevar a cabo su actualización, esto obliga a mantener controlado y documentado el producto. Los cambios de requisitos deben ser gestionados y documentados para asegurar que la calidad de los mismos se mantenga, además de mantener la trazabilidad bidireccional entre la fuente de los requisitos y los productos. Según el libro titulado “Procesos para la Administración de Requisitos” sobre el programa de mejora, la administración de requisitos incluye cuatro subprocesos: Administración de Requisitos, Entendimiento y Compromiso, Traceo y Control de Inconsistencias basado en la meta específica y las cinco prácticas definidas en el área de proceso.

Administración de Requisitos: Durante el proyecto, los requisitos cambian por diversas razones. Es esencial la administración de estos cambios o incorporación de nuevos requisitos de forma eficaz y efectiva. Para un análisis efectivo del impacto de los cambios a los requisitos es necesario que sea conocido el origen de cada requisito y esté documentado el motivo del cambio. Esta gestión de los cambios debe quedar reflejada en la base de datos de requisitos. Tanto las actividades de añadir requisitos como un cambio en ellos debe ser controlado de una forma eficaz y efectiva y deben establecerse las actividades necesarias para controlar el impacto del cambio. (3)

Entendimiento y Compromiso: Alcanzar un entendimiento de los requisitos junto a los productores de los requisitos, generalmente el cliente, acerca de su significado y de este modo evitar un crecimiento de los requisitos fuera del alcance inicial del proyecto. El análisis de los requisitos se hace junto a los proveedores para comprobar que son compatibles y que se comparte el mismo entendimiento y visión de los requisitos por ambas partes. El compromiso trata acerca de los acuerdos entre aquellas personas que deben llevar a cabo las actividades necesarias para implementar los requisitos. Conforme evolucionan y cambian los requisitos, esta práctica asegura que los participantes en el proyecto están comprometidos con los requisitos actuales y aprobados, así como con los cambios resultantes en la planificación y actividades del proyecto. Un cambio en los requisitos es probable que produzca una modificación en la planificación, aumente o disminuya el tiempo planificado, y ambas partes deben estar de acuerdo en estas modificaciones. El compromiso debe obtenerse tanto a nivel externo, con el cliente, como a nivel interno, con el equipo de trabajo. (3)

Traceo: La trazabilidad de requisitos se define como la habilidad para describir y seguir la vida de un requisito en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de todas las especificaciones generadas durante el proceso de desarrollo de software, lo cual garantiza una adecuada administración del cambio con el fin de evaluar el impacto en el resto del sistema.

Control de Inconsistencias: Esta práctica identifica las inconsistencias entre los requisitos establecidos y los planes y productos de trabajo que reflejan esos requisitos e iniciará las acciones correctivas para solucionarlos.

Las actividades relacionadas con la Administración de Requisitos hasta hace poco tiempo se realizaban de forma manual lo que dificultaba y demoraba el trabajo. Con el surgimiento de herramientas modernas para la realización de este proceso se ha disminuido el trabajo duro en el mantenimiento de requisitos, añadiendo beneficios significantes al reducir errores.

1.3 Herramientas para la Administración de Requisitos

El uso de herramientas para auxiliar la gestión de requisitos se ha convertido en un aspecto importante de la Ingeniería de Sistemas y el Diseño. Considerando el tamaño y la complejidad del desarrollo, el uso de herramientas viene siendo algo esencial. Algunas de las herramientas más utilizadas actualmente se presentan a continuación:

- ✓ IBM Rational RequisitePro.

- ✓ Gatherspace.
- ✓ OSRMT.
- ✓ Enterprise Architect.
- ✓ Extensión para la Administración de Requisitos 1.0.

IBM Rational RequisitePro: Es una herramienta de gestión de requisitos y casos prácticos para los equipos de proyecto. Los equipos pueden crear y compartir sus requisitos mediante métodos conocidos basados en documentos, al tiempo que utilizan funciones de la base de datos como la rastreabilidad y el análisis de impacto. De esta manera se mejora la gestión de requisitos y comunicación, se aumenta la calidad y se acelera el tiempo de comercialización. (4)

Ventajas:

- ✓ Evita tareas de remodelación y duplicaciones gracias a la integración avanzada en tiempo real con Microsoft Word.
- ✓ Gestiona la complejidad con vistas de rastreabilidad detalladas que muestran relaciones padre-hijo.
- ✓ Mejora la colaboración de equipos distribuidos geográficamente a través de una interfaz web escalable totalmente funcional e hilos de debate.
- ✓ Captura y analiza información de requisitos con personalización y filtrado detallado de atributos.
- ✓ Ajusta los objetivos empresariales con los productos finales del proyecto mediante la integración con varias herramientas en la plataforma de desarrollo y distribución de software de IBM Rational.

Desventajas:

- ✓ El programa no es multiplataforma y es privativo.
- ✓ No ofrece soporte a pruebas, es necesario utilizar herramientas externas.

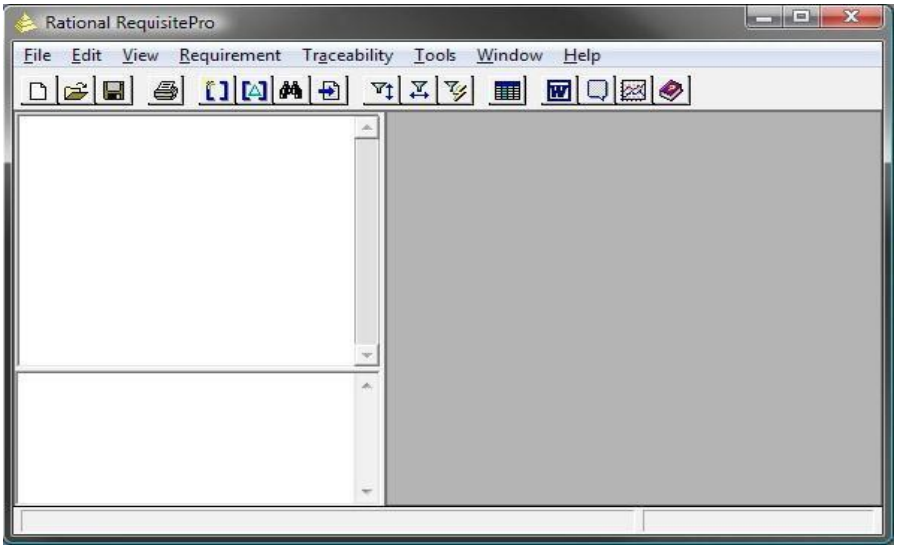


Figura 1: Herramienta de Administración de Requisitos IBM Rational RequisitePro.

Gatherspace: Es una herramienta Web capaz de editar, compartir y distribuir requisitos en línea. Puesto que se necesita únicamente de un navegador Web, esta herramienta se hace independiente de la plataforma, descarta la necesidad de instalación y es portable 100% a cualquier sistema operativo. Es sencilla de utilizar, pues todas las funcionalidades de la aplicación están organizadas en un menú de pestañas muy intuitivo. Los requisitos se agrupan en paquetes y características de manera que están ordenados y son fácilmente localizables dentro de cada paquete. (5)

Ventajas:

- ✓ Permite la gestión de paquetes, características, requisitos de software, iteraciones, actores, casos de uso, crear líneas base o términos en el glosario.
- ✓ Módulo para la gestión de proyectos.
- ✓ Permite crear documentos sobre los requisitos, modelos de casos de uso, trazabilidad entre los diferentes requisitos.
- ✓ Permite buscar paquetes, atributos, actores, casos de uso o cualquier otra característica que se necesite encontrar dentro del proyecto.

Desventajas:

- ✓ Al ser una herramienta web su manejo puede ser lento puesto que está expensa al servidor, el cual puede estar caído o sobrecargado lo que ralentizará o restringirá el trabajo en el proyecto.

- ✓ No dispone de versionado e historial de cambios, que sin ellos la información de los requisitos queda incompleta.
- ✓ Es una herramienta privativa.
- ✓ No provee soporte para la trazabilidad entre requisitos.

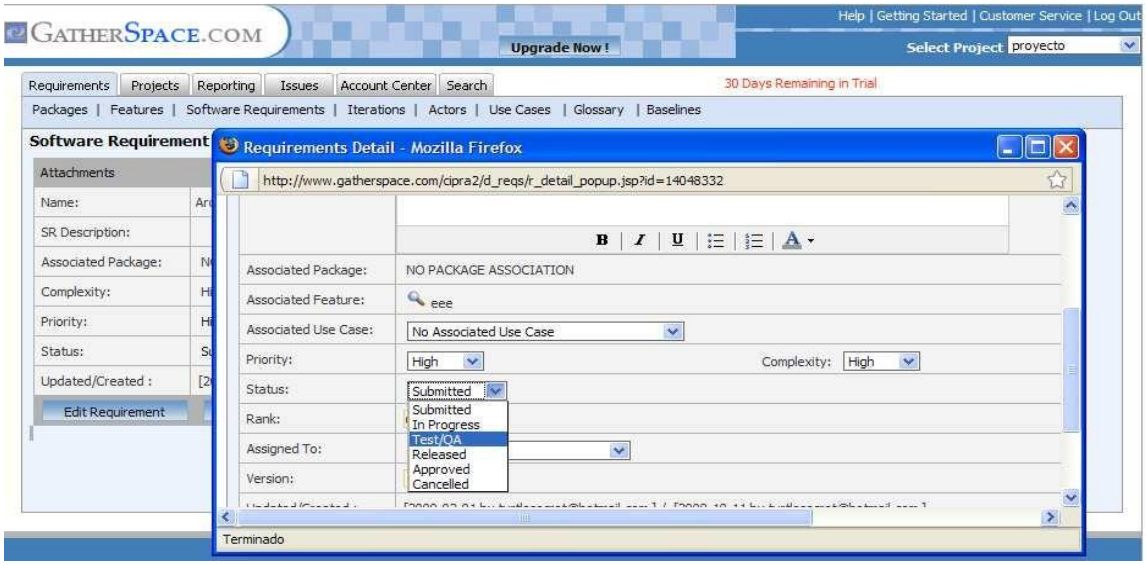


Figura 2: Representación de las características de un requisito en Gatherspace.

OSRMT: Herramienta de software libre que permite una 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 y casos de prueba). (6)

Ventajas:

- ✓ Su licencia es GPL[4].
- ✓ Dispone de búsquedas avanzadas (filtros y ordenaciones) sobre los documentos de trabajo.
- ✓ Es posible visualizar la matriz de trazabilidad entre los diferentes componentes, así como, un árbol de trazabilidad o un gráfico de dependencias entre diferentes documentos.
- ✓ Los informes y estadísticas creados podrán ser básicos o específicos, con la posibilidad de ser exportados a HTML o PDF y la creación de estos informes se hace a través de la herramienta Ireport.
- ✓ Permite la creación de árboles de trazabilidad o matrices de vínculos.

[4] GPL: General Public Licence. Es un tipo de licencia de software.

- ✓ Dentro de las herramientas libres que abarcan la Ingeniería de Requisitos, se trata de la herramienta que mejores funcionalidades ofrece.
- ✓ La visualización de requisitos es intuitiva y fácil de usar.
- ✓ Ofrece un cliente web para accesos desde internet.
- ✓ Posee una amplia documentación.

Desventajas:

- ✓ Consiste en una herramienta llevada a cabo por un único desarrollador por lo que existe un riesgo importante de no poder ser sostenible a largo plazo.
- ✓ Las funcionalidades son escasas, es una herramienta poco madura.
- ✓ La interfaz de usuario es en ocasiones lenta.
- ✓ Algunas funcionalidades no han sido desarrolladas completamente y están a medias.
- ✓ No existe un soporte empresarial.

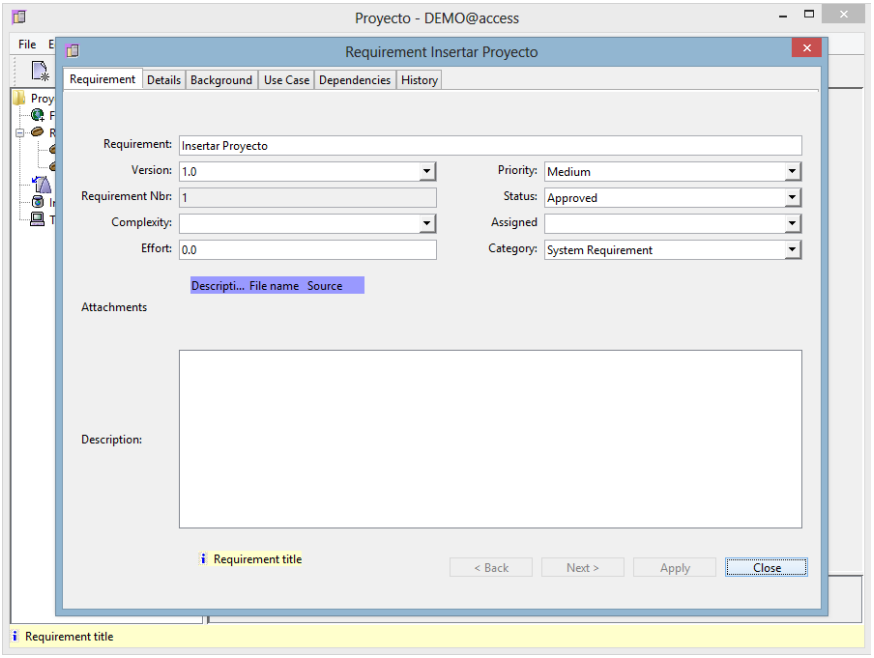


Figura 3: Representación de las características de un requisito en OSRMT.

Enterprise Architect (EA): Enterprise Architect es una herramienta comprensible de diseño y análisis UML (Lenguaje de Modelado Unificado), cubriendo el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. EA es

una herramienta multiusuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece salida de documentación flexible y de alta calidad. (7)

Ventajas:

- ✓ Soporta los 13 diagramas de UML 2.1.
- ✓ Interfaz de usuario intuitiva.
- ✓ Soporte para Transformaciones MDA[5].
- ✓ Documentación flexible y comprensible.
- ✓ Ingeniería de código directa e inversa en varios lenguajes tales como: C++, C#, Delphi, Java, Python, PHP, VB.NET y clases de Visual Basic.
- ✓ Soporte en Administración de Requisitos.

Desventajas:

- ✓ Es una herramienta privativa.
- ✓ Los diagramas no se encuentran a la vista.

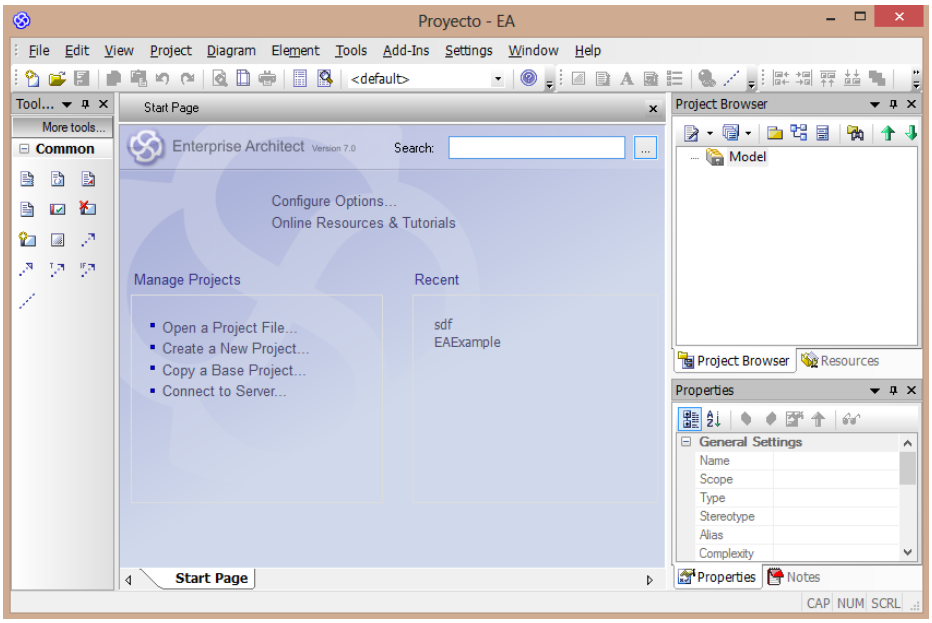


Figura 4: Herramienta CASE para la Administración de Requisitos Enterprise Architect.

[5] MDA: Model Driven Architecture. Es un nuevo paradigma que concibe la construcción de modelos de software, a distintos niveles de abstracción, como artefactos principales en el desarrollo de software.

Extensión para la Administración de Requisitos (VP_REQM) 1.0: Es una extensión de la herramienta CASE "*Visual Paradigm for UML*", de software libre, desarrollada en el departamento de Integración de Soluciones para la Administración de Requisitos con el objetivo de automatizar dicho proceso.

Ventajas:

- ✓ Permite la gestión de proyectos, módulos, requisitos y elementos de la base de datos de cada proyecto.
- ✓ Brinda la posibilidad de generar la matriz de trazabilidad.
- ✓ Posibilita interpretar diagramas modelados en la herramienta.
- ✓ Tiene soporte para la generación de reportes en formato PDF y XLS.
- ✓ Se cuenta con el código fuente de la misma ya que fue realizada en el centro.

Desventajas:

- ✓ Garantiza parcialmente la trazabilidad bidireccional con los elementos predefinidos a tracear dentro del proyecto, definidos en la guía de trazabilidad del proceso de mejora.
- ✓ Limitantes en cuanto a la seguridad y validaciones en funcionalidades desarrolladas.
- ✓ Problemas de usabilidad.

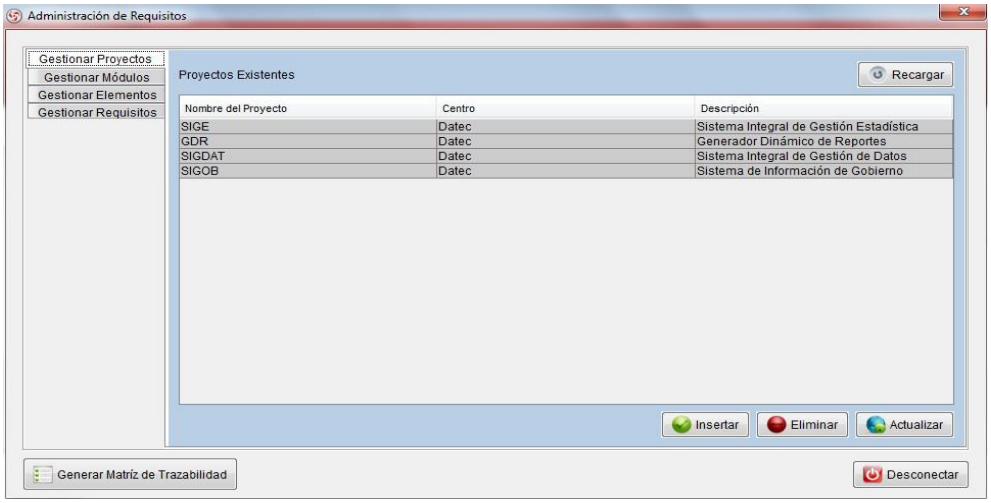


Figura 5: Extensión para la Administración de Requisitos versión 1.0.

A continuación se presenta una tabla comparativa representando las características principales de cada una de estas herramientas:

Herramientas	Características			
	Privativa	Portable	Multiplataforma	Soporte
Enterprise Architect	Sí	No	Sí	Sí
Gatherspace	Sí	No	Sí	Sí
RequisitePro	Sí	No	No	Sí
OSRMT	No	Sí	Sí	No
VP_REQM 1.0.	No	Sí	Sí	Sí

Tabla 1: Tabla comparativa entre herramientas para la Administración de Requisitos.

Después de un estudio de las herramientas y de conocer las funcionalidades que cada una ofrece, se puede identificar que la extensión para la Administración de Requisitos utilizada en el departamento de Integración de Soluciones es la más viable para este proceso, pues es una herramienta de software libre, multiplataforma, se le pueden agregar nuevas funcionalidades y las que posee pueden mejorarse, pues es una herramienta desarrollada en el centro. Por todo lo antes expuesto se propone realizar la versión 2.0 de la herramienta en aras de garantizar una correcta administración de los requisitos en el proceso de desarrollo de software.

1.4 Plataforma tecnológica

Metodología de desarrollo de software.

Una metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado. Es un proceso de software detallado y completo. Existen varias metodologías a seguir para el desarrollo de software en la actualidad, estas se dividen en dos categorías metodologías ágiles y metodologías robustas. Las robustas centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto. Las metodologías ágiles nacen como respuesta a los problemas que puedan ocasionar las metodologías robustas y se basa en dos aspectos fundamentales, retrasar las decisiones y la planificación adaptativa. Basan su fundamento en la adaptabilidad de los procesos de desarrollo. (8) Entre las metodología ágiles se encuentra OpenUP.

OpenUP

Constituye un marco de trabajo para procesos de desarrollo de software. Fue liberado por el Eclipse Process Framework (EPF). Se construyó sobre una donación realizada por IBM del Basic Unified Process. Fue entregada a Eclipse a fines de 2005 y renombrado como OpenUP en 2006. Es un proceso de desarrollo unificado que está basado en Rational Unified Process (RUP) y forma parte de la familia del Proceso Unificado que lo conforman además de los mencionados Agile Unified Process (AUP), Enterprise Unified Process (EUP), Basic Unified Process (BUP). El ciclo de vida de un proyecto según esta metodología se divide en 4 fases fundamentales: Concepción, Elaboración, Construcción y Transición. Propone 6 flujos de trabajo: Requisitos, Arquitectura, Desarrollo, Prueba, Gestión del Proyecto y Gestión de Configuración y Cambios. (9) Después de realizar un análisis de la metodología se seleccionó OpenUP por las siguientes características:

- ✓ Es la metodología utilizada por el departamento.
- ✓ Es la que más se acoge a las necesidades del equipo de desarrollo.
- ✓ Es un proceso ágil de desarrollo del software.
- ✓ Es extensible ya que en el proceso se pueda agregar o adaptar según lo vayan requiriendo los sistemas.
- ✓ Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.

Lenguaje de modelado

UML 2.0: Lenguaje de Modelado Unificado, se centra en la representación gráfica de un sistema. Está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware. (10)

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones (10)**:**

- ✓ **Visualizar:** UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- ✓ **Especificar:** UML permite especificar cuáles son las características de un sistema antes de su construcción.

- ✓ **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- ✓ **Documentar:** Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Lenguaje de programación

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo. (11)

Java

Es un lenguaje de programación de alto nivel, orientado a objetos, sencillo, independiente de plataforma, con gran nivel de seguridad. Fue creado por James Gosling, de Sun Microsystems, en 1990. Presenta las siguientes características (12):

- ✓ Simple: Elimina la complejidad de los lenguajes como "C" y da paso al contexto de los lenguajes modernos orientados a objetos.
- ✓ Robusto: El sistema de Java maneja la memoria de la computadora. Excluyendo al programador de preocuparse por apuntadores y de la memoria que no se esté utilizando.
- ✓ Seguro: El sistema de Java tiene ciertas políticas que evitan se puedan codificar virus con este lenguaje. Existen muchas restricciones, especialmente para los applets[6], que limitan lo que se puede y no puede hacer con los recursos críticos de una computadora.
- ✓ 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.
- ✓ Independiente a la arquitectura: Al compilar un programa en Java, el código resultante es un tipo de código binario conocido como *byte code*. Este código es interpretado por diferentes computadoras de igual manera, solamente hay que implementar un intérprete para cada plataforma. De esa manera Java logra ser un lenguaje que no depende de una arquitectura computacional definida.
- ✓ Multitarea: Ejecuta varias tareas o subprocesos simultáneamente.
- ✓ Interpretado: Java corre en máquina virtual, por lo tanto es interpretado.
- ✓ Dinámico: No requiere compilar todas las clases de un programa para que este funcione.

[6]Applets: Componente de software que suele ser pequeño escrito en un lenguaje de programación.

El lenguaje Java brinda facilidades para generar reportes en diferentes formatos lo que va a posibilitar ver a cierto nivel de detalle la información o parte de la misma.

El lenguaje de programación en cuestión contiene bibliotecas para crear dinámicamente documentos PDF, WORD, XLS, ayudando a la generación de reportes. Con estas bibliotecas se puede insertar textos, tablas, gráficos, además contar con diferentes formas de visualización, permiten la utilización de diferentes tipologías de letras, facilitan la organización del documento por secciones y capítulos. Algunas de estas bibliotecas son iText y jxl que permiten generar reportes en PDF y Excel respectivamente, las cuales serán utilizadas en la extensión.

Este lenguaje también consta de bibliotecas para realizar pruebas a las aplicaciones ayudando a verificar el correcto funcionamiento de las mismas. Una de estas bibliotecas es JUnit que permite realizar pruebas unitarias mediante la ejecución controlada a las clases, lo que permite comprobar el funcionamiento de cada uno de los métodos, pues en función de algunos valores de entrada se evalúa el resultado esperado. Esta biblioteca será utilizada para automatizar el proceso de pruebas unitarias a la extensión.

Por las características mencionadas anteriormente del lenguaje de programación Java, además de que es el lenguaje propuesto por la API[7] de desarrollo de la herramienta "*Visual Paradigm for UML*", y en la actualidad es uno de los lenguajes más utilizados a nivel mundial por su robustez y seguridad a la hora de desarrollar aplicaciones, se decide utilizar como lenguaje de desarrollo.

Herramientas a utilizar.

Un IDE (Integrated Development Environment) es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. (13)

[7]API: Interfaz de Programación de Aplicaciones. Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Netbeans IDE 7.1

Es un entorno de desarrollo una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso. (14) Este proporciona un rendimiento superior para los lenguajes C / C++ y los desarrolladores de PHP, proporcionando editores y herramientas integrales para sus marcos y tecnologías relacionadas. Además, el IDE tiene editores y herramientas para XML, HTML, PHP, Groovy, Javadoc, JavaScript y JSP.

Se asume para el desarrollo de la extensión el Netbeans 7.1 por las comodidades que brinda además de ser gratuito, de código abierto y tener una gran comunidad de usuarios y desarrolladores de todo el mundo.

Herramientas CASE

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 pueden ayudar 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 costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. (15)

Visual Paradigm for UML 8.0 (VP-UML)

Es una herramienta de diseño UML y herramienta CASE UML diseñada para ayudar al desarrollo de software. VP-UML soporta los principales estándares de la industria tales como el Lenguaje de Modelado Unificado (UML), SysML[8], BPMN[9], XMI[10], ofrece un completo conjunto de herramientas de equipos de desarrollo de software necesario para la captura de requisitos, software de planificación, la planificación de controles, el modelado de clases, modelado de datos. (16)

[8]SysML: System Modeling Language. Es un lenguaje de especificación de sistemas.
[9] BPMN: Business Process Modeling Notation.
[10]XMI: XML Metadata Interchange. Es una especificación para el Intercambio de Diagramas.

Las principales características de la herramienta son:

- ✓ Soporta las últimas versiones del Lenguaje de Modelado Unificado (UML) y la Notación y Modelado de Procesos del Negocio.
- ✓ Provee un generador de mapeo de objetos relacionales para los lenguajes de programación Java, PHP y para la plataforma .Net.
- ✓ Posee integración con diferentes herramientas como Eclipse, JBuilder, Netbeans IDE, Oracle JDeveloper y BEA Weblogic.
- ✓ Posee un poderoso generador de documentación y reportes en formato PDF, HTML y MS Word.
- ✓ Proporciona soporte para varios lenguajes en la Generación de Código e Ingeniería Inversa como: Java, C++, CORBA IDL, PHP, Ada y Python.
- ✓ Importación y exportación de ficheros.

Se utilizará esta herramienta para el modelado por todas las características antes expuestas, pues a pesar de ser un software propietario, la universidad cuenta con su licencia. Además es la herramienta que utiliza el centro DATEC en sus producciones, es multiplataforma y soporta todos los diagramas UML.

ORM

Mapeo objeto-relacional, (más conocido por su nombre en inglés, Object-Relational Mapping, o sus siglas O/RM, ORM, y O/R mapping). Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos. En la práctica esto crea una base de datos orientada a objetos virtual, por sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). (17)

Se realizó un estudio de los ORM que propone el lenguaje de programación seleccionado destacándose entre estos Hibernate, JPA y ORMLite como los más utilizados. Para trabajar con Hibernate existen dos formas, una de ellas es utilizando XML y la otra es con anotaciones, en ambos casos posee problemas en cuanto a que no realiza las funciones necesarias para la conexión y la gestión de datos, pues se pierde en la llamada a un método propio de una de sus bibliotecas al integrarse con el "*Visual Paradigm for UML*". JPA no se ajusta a la estructura de la extensión, pues para funcionar necesita un XML que se encuentra dentro de una carpeta que va en la raíz del proyecto y modifica por tanto la estructura de la extensión por

lo que no se puede utilizar. El más viable es ORMLite que es el único que se ajusta completamente a la extensión, proporcionando todas las funcionalidades requeridas.

ORMLite

Object Relational Mapping Lite proporciona algunas funcionalidades de peso ligero para la persistencia de objetos Java a bases de datos SQL, evitando la complejidad y la sobrecarga de más paquetes ORM estándar.

ORMLite ofrece las siguientes características (18):

- ✓ Configuración de sus clases por simple adición de anotaciones Java.
- ✓ Potentes clases abstractas para el acceso a datos mediante DAO (Objeto de Acceso a Datos).
- ✓ Generador de consultas flexibles para construir fácilmente consultas simples y complejas.
- ✓ Soporta MySQL, PostgreSQL, Microsoft SQL Server, H2, Derby, HSQLDB, y SQLite y se puede extender a otras bases de datos con relativa facilidad.
- ✓ Soporte básico para las transacciones de base de datos.
- ✓ Auto genera SQL para crear y eliminar tablas de bases de datos.
- ✓ Soporte para configuración de Spring a través de DAOs y configuraciones de clases.

Se decide utilizar ORMLite además de las características expuestas anteriormente porque es el único de los estudiados que se ajusta correctamente a la estructura de la extensión, es un ORM ligero, rápido y de una gran facilidad de uso.

1.5 Conclusiones parciales del capítulo.

Luego de realizado el estudio de la Ingeniería de Requisitos, sus principales procesos y herramientas informáticas que realizan el proceso de Administración de Requisitos, así como el análisis de la plataforma tecnológica o ambiente de trabajo para el desarrollo de la solución se concluye lo siguiente:

- ✓ Desarrollar la versión 2.0 de la extensión para la Administración de Requisitos.
- ✓ Se utilizará como lenguaje de programación Java y como IDE de desarrollo NetBeans 7.1
- ✓ Para el modelado se utilizará la herramienta CASE Visual Paradigm for UML 8.0, junto al lenguaje de modelado UML 2.0 siguiendo la metodología de desarrollo de software OpenUP
- ✓ Para migrar el código de acceso a datos para un ORM (Mapeo Relacional de Objetos) soportado por el lenguaje de programación se utilizará ORMLite.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA EXTENSIÓN

En el presente capítulo se describen los conceptos más importantes del 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. 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 se dan a conocer los diferentes tipos de diagramas que representan la lógica y el funcionamiento del sistema, además se muestran los patrones de diseño que se aplican en la extensión.

2.1 Modelo de dominio

Un modelo del dominio es una representación visual de las clases conceptuales u objetos del mundo real en un entorno de interés, captura los tipos más importantes de objetos en el contexto del sistema. Muchos de los objetos del dominio o clases pueden obtenerse de una especificación de requisitos o mediante la entrevista con los expertos del medio. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del medio y para lograr una mayor comprensión se describen los objetos del dominio. (19)

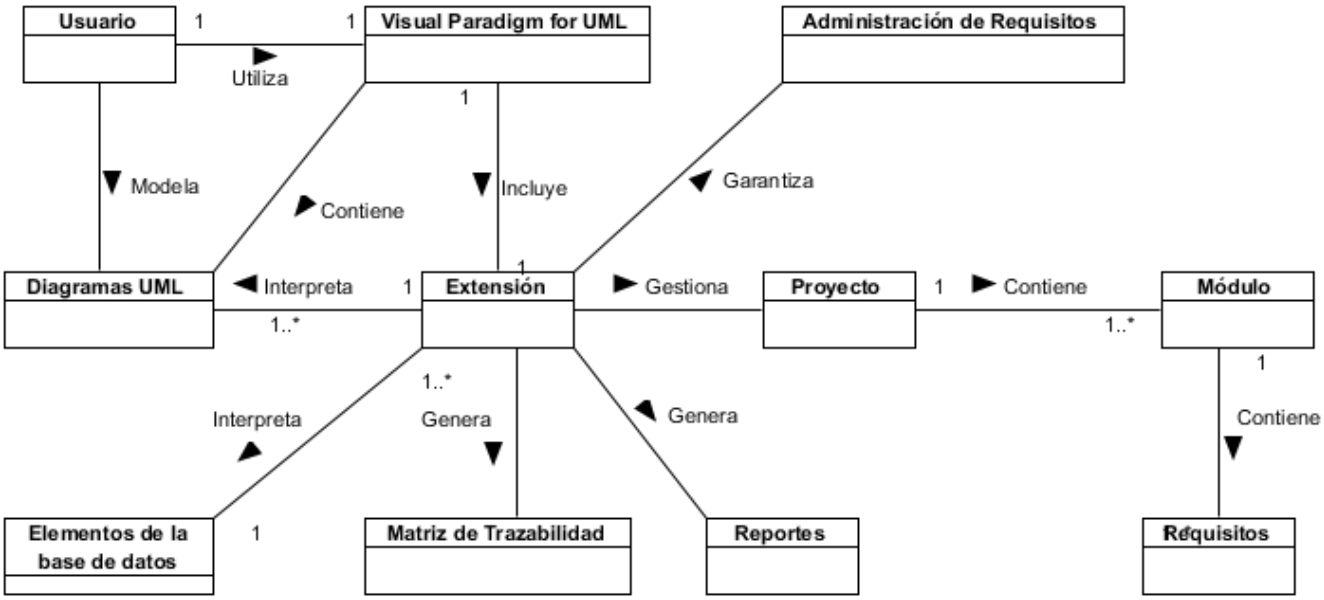


Figura 6: Modelo de Dominio.

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, encargada de modelar y contener los diagramas para la posterior interpretación.

Diagramas UML: Representación gráfica en la que se muestran los diagramas, los cuales son interpretados por la extensión.

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.

Extensión: Representa la extensión en su totalidad la cual se encarga de interpretar los diferentes tipos de diagramas que se construyen en la herramienta "*Visual Paradigm for UML*", los elementos de la Base de Datos, es capaz de gestionar además proyectos, módulos y requisitos para posteriormente realizar la trazabilidad con los requisitos generando la matriz de trazabilidad y permitiendo la generación de reportes en diferentes formatos.

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.

Matriz de trazabilidad: Representa la matriz obtenida después de realizar el proceso de trazabilidad entre los elementos seleccionados por el usuario.

Reportes: Representa el reporte con la información requerida por el usuario obtenido después de realizar el proceso de trazabilidad.

Proyecto: Representa la clase entidad que contiene todos los datos del proyecto, incluyendo los módulos asignados al mismo.

Módulo: Representa la clase entidad que contiene todos los datos del módulo y los requisitos asignados al mismo.

Requisitos: Representa la clase entidad que contiene los datos del requisito.

2.2 Especificación de los requisitos de la extensión

Se identificaron los siguientes requisitos funcionales:

RF_1 Establecer conexión con la base de datos.	**Descripción:** El sistema debe permitir establecer la conexión con la base de datos. **Entrada:** Datos de la conexión (usuario, contraseña, servidor, host, puerto). **Salida:** Conexión realizada.
Gestionar proyecto.	
RF_2 Insertar proyecto.	**Descripción:** El sistema debe permitir insertar un nuevo proyecto. **Entrada:** Se ingresan los datos (nombre del proyecto, centro y descripción). **Salida:** Mensaje de confirmación de la inserción.
RF_3 Actualizar proyecto.	**Descripción:** el sistema debe permitir actualizar un proyecto. **Entrada:** Campos para la actualización (nombre del proyecto, centro, descripción). **Salida:** Proyecto actualizado.
RF_4 Eliminar proyecto.	**Descripción:** El sistema debe permitir eliminar los proyectos. **Entrada:** La lista con los proyectos a eliminar. **Salida:** Proyectos eliminados.
RF_5 Listar proyecto.	**Descripción:** El sistema debe permitir listar los proyectos existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los proyectos. **Salida:** Listado de los proyectos encontrados.
Gestionar módulo.	
RF_6 Insertar módulo.	**Descripción:** El sistema debe permitir insertar un nuevo módulo. **Entrada:** Se ingresan los datos (nombre del módulo, proyecto al que pertenece). **Salida:** Módulo insertado.
RF_7 Actualizar módulo.	**Descripción:** El sistema debe permitir actualizar los datos del módulo. **Entrada:** Campos para la actualización (nombre del módulo, proyecto al que pertenece). **Salida:** Elemento actualizado.
RF_8 Eliminar módulo.	**Descripción:** El sistema debe permitir eliminar los módulos.

	Entrada: La lista con los módulos a eliminar. **Salida:** Módulos eliminados.
RF_9 Listar módulo.	**Descripción:** El sistema debe permitir listar los módulos existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los módulos. **Salida:** Listado de los módulos encontrados.
Gestionar elemento	
RF_10 Insertar elemento.	**Descripción:** El sistema debe permitir insertar un nuevo elemento **Entrada:** Se ingresan los datos (nombre, tipo de elemento). **Salida:** Nuevo elemento insertado.
RF_11 Actualizar elemento.	**Descripción:** El sistema debe permitir actualizar los datos del elemento. **Entrada:** Campos para la actualización (nombre, tipo de elemento). **Salida:** Elemento actualizado.
RF_12 Eliminar elemento.	**Descripción:** El sistema debe permitir eliminar los elementos. **Entrada:** La lista con los elementos a eliminar. **Salida:** Elementos eliminados.
RF_13 Listar elemento.	**Descripción:** El sistema debe permitir listar los elementos existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los elementos de la base de datos. **Salida:** Listado de los elementos encontrados.
Gestionar requisito.	
RF_14 Insertar requisito.	**Descripción:** El sistema debe permitir insertar un nuevo requisito. **Entrada:** Se ingresan los datos (módulo, estado, nombre, clasificación, descripción, prioridad, complejidad, autor, discriminante). **Salida:** Nuevo requisito insertado.
RF_15 Actualizar requisito.	**Descripción:** El sistema debe permitir actualizar los datos de los requisitos. **Entrada:** Campos para la actualización (nombre, módulo, proyecto). **Salida:** Mensaje de confirmación de la actualización.
RF_16 Eliminar requisito.	**Descripción:** Se selecciona cual requisito se desea eliminar los requisitos.

	Entrada: La lista con los requisitos a eliminar. **Salida:** Requisitos eliminados.
RF_17 Listar requisito.	**Descripción:** El sistema debe permitir listar los requisitos existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los requisitos. **Salida:** Listado de los requisitos encontrados.
Gestionar usuario.	
RF_18 Insertar usuario.	**Descripción:** El sistema debe permitir insertar un nuevo usuario. **Entrada:** Se ingresan los datos (servidor LDAP, nombre, contraseña, dominio). **Salida:** Nuevo usuario insertado.
RF_19 Actualizar usuario.	**Descripción:** El sistema debe permitir actualizar los datos de los usuarios. **Entrada:** Campos para la actualización (servidor LDAP, nombre, contraseña, dominio). **Salida:** Usuario actualizado.
RF_20 Eliminar usuario.	**Descripción:** El sistema debe permitir eliminar los usuarios. **Entrada:** La lista con los usuarios a eliminar. **Salida:** Usuarios eliminados.
RF_21 Listar usuario.	**Descripción:** El sistema debe permitir listar los usuarios existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los usuarios. **Salida:** Listado de los usuarios encontrados.
Gestionar LDAP.	
RF_22 Insertar LDAP.	**Descripción:** El sistema debe permitir insertar un nuevo servidor LDAP. **Entrada:** Se ingresan los datos (nombre, servidor, usuario, contraseña, dominio, campo de usuario). **Salida:** nuevo servidor LDAP insertado.
RF_23 Actualizar LDAP.	**Descripción:** El sistema debe permitir actualizar los datos de los servidores LDAP. **Entrada:** Campos para la actualización (nombre, servidor, usuario, contraseña, dominio, campo usuario). **Salida:** Servidor LDAP actualizado.
RF_24 Eliminar LDAP.	**Descripción:** El sistema debe permitir eliminar los servidores LDAP. **Entrada:** La lista con los servidores LDAP a eliminar.

	Salida: Servidores LDAP eliminados.
RF_25 Listar LDAP.	**Descripción:** El sistema debe permitir buscar los servidores LDAP existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los servidores LDAP. **Salida:** Listado de los servidores LDAP encontrados.
Gestionar Diagrama.	
RF_26 Insertar Diagrama.	**Descripción:** El sistema debe permitir insertar un nuevo diagrama. **Entrada:** Se ingresan los datos (nombre, tipo). **Salida:** Nuevo diagrama insertado.
RF_27 Actualizar Diagrama.	**Descripción:** El sistema debe permitir actualizar los datos de los diagramas. **Entrada:** Campos para la actualización (nombre, tipo). **Salida:** Diagrama actualizado.
RF_28 Eliminar Diagrama.	**Descripción:** El sistema debe permitir eliminar un diagrama. **Entrada:** La lista con los diagramas a eliminar. **Salida:** Diagramas eliminados.
RF_29 Listar Diagrama.	**Descripción:** El sistema debe permitir listar los diagramas existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los diagramas. **Salida:** Listado de los diagramas encontrados.
Gestionar Componente.	
RF_30 Insertar Componente.	**Descripción:** El sistema debe permitir insertar un nuevo componente. **Entrada:** Se ingresan los datos (nombre, tipo). **Salida:** Nuevo componente insertado.
RF_31 Actualizar Componente.	**Descripción:** El sistema debe permitir actualizar los datos de los componentes. **Entrada:** Campos para la actualización (nombre, tipo). **Salida:** Componente actualizado.
RF_32 Eliminar Componente.	**Descripción:** El sistema debe permitir eliminar los componentes. **Entrada:** La lista con los componentes a eliminar. **Salida:** Componentes eliminados.
RF_33 Listar Componente.	**Descripción:** El sistema debe permitir listar los componentes existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los componentes. **Salida:** Listado de los componentes encontrados.

Administrar tipo de elemento.	
RF_34 Insertar tipo de elemento.	**Descripción:** El sistema debe permitir insertar un nuevo tipo de elemento. **Entrada:** Se ingresan los datos (nombre, relación). **Salida:** Nuevo tipo de elemento insertado.
RF_35 Eliminar tipo de elemento.	**Descripción:** El sistema debe permitir eliminar los tipos de elementos. **Entrada:** Tipo de elemento a eliminar. **Salida:** Tipo de elemento eliminado.
RF_36 Listar tipo de elemento.	**Descripción:** el sistema debe permitir listar los tipos de elementos existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los tipos de elementos. **Salida:** Listado de los tipos de elementos encontrados.
Administrar estado.	
RF_37 Insertar estado.	**Descripción:** Se ingresan los datos para la inserción. **Entrada:** Se ingresan los datos del nuevo estado (nombre). **Salida:** Nuevo estado insertado.
RF_38 Eliminar estado.	**Descripción:** El sistema debe permitir eliminar los estados existentes en la base de datos. **Entrada:** Estado a eliminar. **Salida:** Estado eliminado.
RF_39 Listar estado.	**Descripción:** El sistema debe permitir listar los estados existentes en la base de datos. **Entrada:** Se realiza una búsqueda de los estados. **Salida:** Listado de los estados encontrados.
Administrar clasificación.	
RF_40 Insertar clasificación.	**Descripción:** Se ingresan los datos para la inserción. **Entrada:** Se ingresan los datos de la nueva clasificación (nombre). **Salida:** Nueva clasificación insertada.
RF_41 Listar clasificación.	**Descripción:** el sistema debe permitir listar las clasificaciones existentes en la base de datos. **Entrada:** Se realiza una búsqueda de las clasificaciones. **Salida:** Listado de las clasificaciones encontradas.
RF_42 Eliminar clasificación.	**Descripción:** El sistema debe permitir eliminar las clasificaciones. **Entrada:** Clasificación a eliminar. **Salida:** Clasificación eliminada.

RF_43 Buscar	**Descripción:** El sistema debe permitir buscar usuarios, proyectos, módulos, requisitos, elementos y servidores LDAP existentes en la base de datos. **Entrada:** Criterio de búsqueda. **Salida:** Usuarios, proyectos, módulos, requisitos, elementos o servidores LDAP.
RF_44 Generar Matriz de Trazabilidad.	**Descripción:** El sistema debe permitir generar la matriz de trazabilidad. **Entrada:** Listado con los elementos para realizar la trazabilidad. **Salida:** Matriz de trazabilidad.
RF_45 Generar Árbol de Trazabilidad.	**Descripción:** El sistema debe permitir generar el árbol de trazabilidad. **Entrada:** Listado con los elementos para realizar la trazabilidad. **Salida:** Árbol de trazabilidad.
RF_46 Generar reportes.	**Descripción:** El sistema debe permitir generar reportes. **Entrada:** Opciones de reportes disponibles. **Salida:** Reporte en el formato seleccionado con la información correspondiente.
RF_47 Asignar Módulo.	**Descripción:** El sistema debe permitir asignar un módulo. **Entrada:** Se selecciona el módulo a asignar. **Salida:** Módulo asignado.
RF_48 Revocar Módulo.	**Descripción:** El sistema debe permitir revocar un módulo. **Entrada:** Se selecciona el módulo a revocar. **Salida:** Módulo revocado.
RF_49 Cambiar contraseña.	**Descripción:** El sistema debe permitir cambiar la contraseña. **Entrada:** Se ingresa la nueva contraseña (contraseña anterior, nueva contraseña, confirmación). **Salida:** Contraseña cambiada.
RF_50 Autenticar usuario.	**Descripción:** El sistema debe permitir la autenticación. **Entrada:** Se ingresa los datos de autenticación (usuario, contraseña) **Salida:** Usuario autenticado.

Requisitos no funcionales de la extensión.

Usabilidad

- ✓ ***RNF_1*** La herramienta deberá presentar facilidades al usuario para el manejo de la información, se pretende una vista descriptiva para la realización de todas las operaciones con el objetivo de propiciar un buen entendimiento de la herramienta a los clientes finales.
- ✓ ***RNF_2*** La herramienta deberá presentar mensajes para mantener informado al usuario de las operaciones realizadas.

Restricciones del diseño e implementación

- ✓ ***RNF_3*** Se utilizarán para la realización del producto, la herramienta de modelado "*Visual Paradigm for UML*" en su versión 8.0 y como entorno de desarrollo NetBeans 7.1
- ✓ ***RNF_4*** El lenguaje de programación que será empleado para la implementación será Java utilizando el JDK 1.7 (Java Development Kit) siguiendo el paradigma de la Programación Orientada a Objeto y su compatibilidad con otros sistemas operativos.
- ✓ ***RNF_5*** Se utilizarán como Sistemas Gestores de Base de Datos PostgreSQL, MySQL o SQLite.

Seguridad

- ✓ ***RNF_6*** El usuario deberá autenticarse en la herramienta para el manejo de la información almacenada en la base de datos.

Hardware

- ✓ ***RNF_7*** Mínimo 512 MB de RAM (Random Access Memory, por sus siglas en inglés), pero se recomienda 1GB.
- ✓ ***RNF_8*** Un mínimo de 400 MB de espacio en disco.

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 (CU). 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 describe la funcionalidad que

se construirá en el sistema propuesto. Un caso de uso puede "incluir" la funcionalidad de otro caso de uso o "extender" a otro caso de uso con su propio comportamiento. (20)

Actor del sistema

Un actor es un usuario del sistema. Incluye usuarios humanos y otros sistemas computarizados. Un actor usa un caso de uso para desempeñar alguna porción de trabajo que es de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define su rol global en el sistema y el alcance de su acción. (20)

En el modelo de casos de uso del sistema se identificaron los siguientes actores.

Nombre del Actor	Descripción
Usuario	Es el encargado de realizar la gestión de los elementos y de los requisitos dentro de la extensión para poder realizar una correcta administración de requisitos.
Administrador	Es el encargado de gestionar los usuarios, proyectos módulos, requisitos, elementos de la base de datos y servidores LDAP dentro de la extensión garantizando un correcto funcionamiento de la misma y posibilitando la correcta administración de requisitos.

Tabla 2: Descripción de los Actores del Sistema.

Diagrama de Casos de Uso del Sistema.

Los diagramas de casos de uso (DCUS) documentan el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto los casos de uso determinan los requisitos funcionales del sistema, es decir, representan las funciones que un sistema puede ejecutar.

A continuación se muestra el diagrama de casos de uso del sistema, el mismo posee 18 casos de uso, los cuales fueron agrupados utilizando los patrones de casos de uso: extensión concreta, CRUD: completo y parcial sobre los requisitos funcionales.

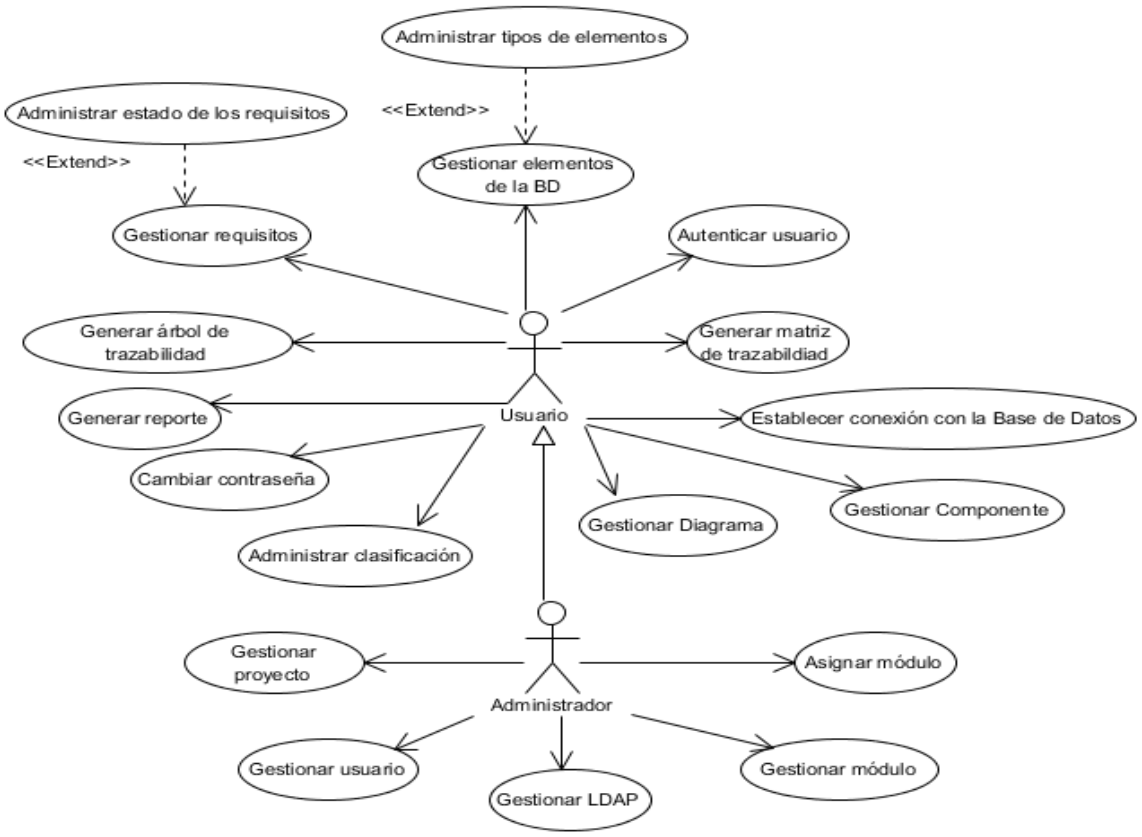


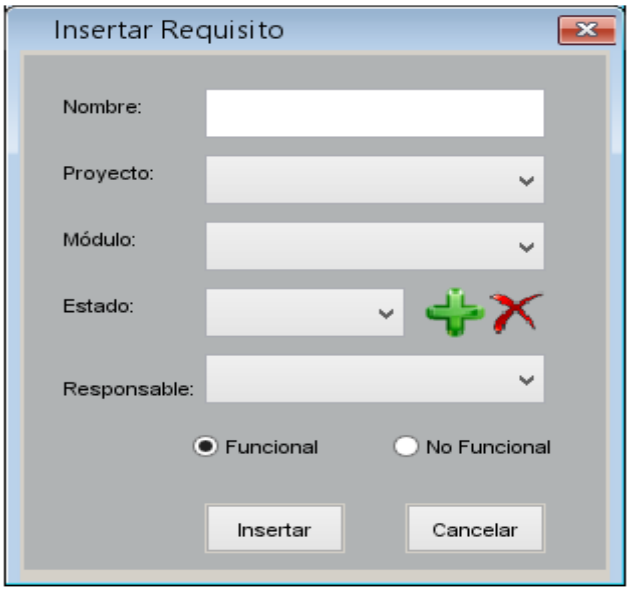
Figura 7: Diagrama de Casos de Uso del Sistema.

Descripción textual del Caso de Uso Gestionar Requisito.

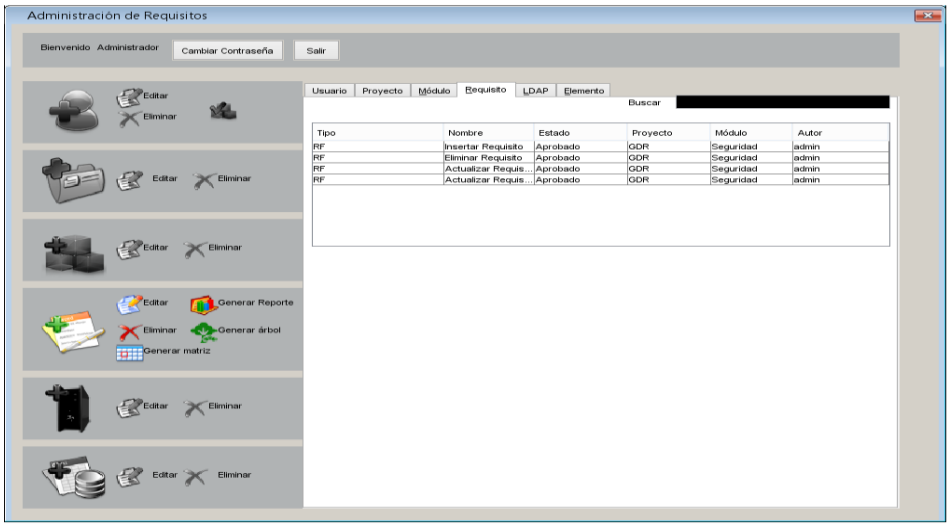
Objetivo	Gestionar todos los requisitos que se llevarán a cabo a lo largo del trabajo en la extensión.
Actores	Usuario
Resumen	El usuario que está trabajando en la extensión elige gestionar requisitos en la interfaz principal desde donde podrá insertar, buscar, eliminar y actualizar cualquier requisito.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	Debe el usuario estar autenticado y existir al menos un proyecto y un módulo creado.

Poscondiciones	Gestión de un requisito.	
Flujo de eventos		
	Actor	**Sistema**
1	Selecciona en la interfaz principal la opción requisitos.	
2		El sistema muestra la interfaz de gestionar requisitos donde el usuario tiene la posibilidad de escoger la acción que desea trabajar. Opción Adicionar ver sección "Insertar Requisito" Opción Eliminar ver sección "Eliminar Requisito" Opción Editar ver sección "Actualizar Requisito"
3		Termina el caso de uso.
Sección: "Insertar Requisito"		
	Actor	**Sistema**
1.	Selecciona la opción "Adicionar"	Visualiza la interfaz Insertar Requisito.
2.	Introduce los datos para el nuevo requisito. ✓ Nombre ✓ Proyecto ✓ Módulo ✓ Estado ✓ Responsable ✓ Funcional o No funcional (Selección)	

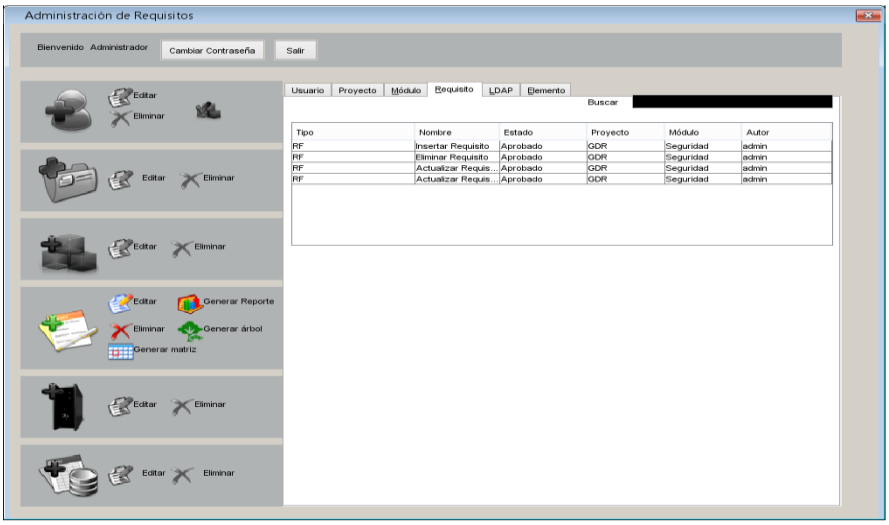
	Selecciona la opción No funcional ir al paso 7	
3		Muestra la interfaz Requisito Funcional con la prioridad, complejidad y las dependencias.
4	Selecciona la prioridad, complejidad y las dependencias y elige la opción Aceptar.	Cierra la interfaz Requisito Funcional y visualiza la opción Insertar en la interfaz Insertar Requisito.
5	Selecciona la opción Insertar	Valida que los datos introducidos estén correctamente, si son incorrectos ir al paso1 del flujo alterno.
6		Muestra un mensaje de confirmación.
7		Muestra la interfaz Requisito No Funcional con la clasificación y descripción.
8	Selecciona la opción Insertar	Valida que los datos introducidos estén correctamente, si son incorrectos ir al paso 1 del flujo alterno.
9		Muestra un mensaje de confirmación.
	Prototipo de Interfaz	



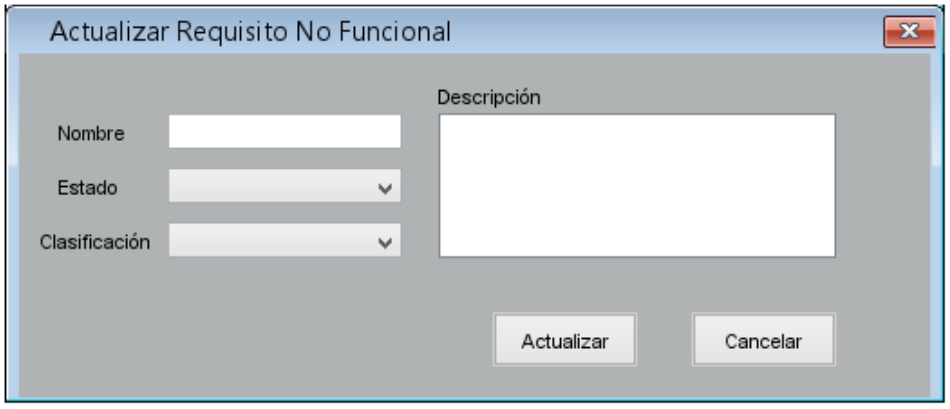
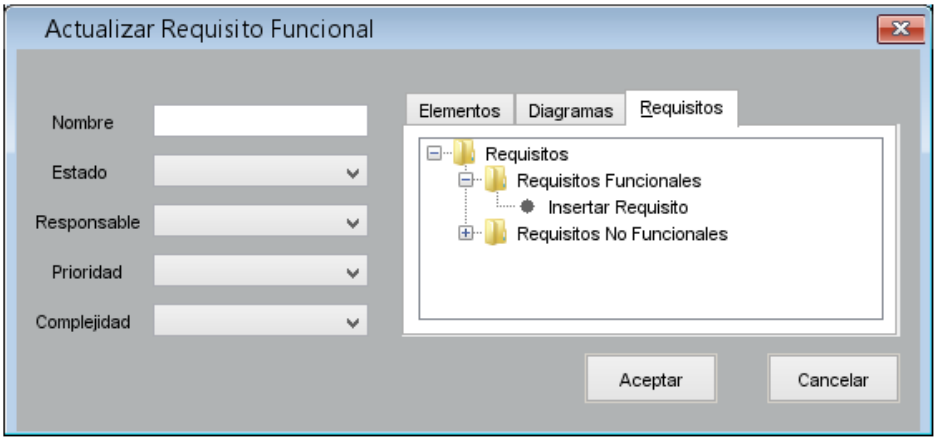
Flujos alternos		
	Actor	**Sistema**
1.		Muestra un mensaje de error para indicarle al usuario que no puede insertar el requisito.
Sección: "Eliminar Requisito"		
	Actor	**Sistema**
1.	Selecciona uno o varios requisitos.	Verifica que haya un requisito seleccionado, sino existe ninguno seleccionado ver paso 1 del flujo alterno.
2.	Selecciona la opción Eliminar.	Muestra un mensaje de confirmación para la eliminación.
3.	El usuario confirma la opción.	
	Prototipo de Interfaz	



Flujos alternos		
	Actor	**Sistema**
1		Muestra mensaje para indicarle al usuario que debe seleccionar un requisito.
Sección: "Listar Requisito"		
	Actor	**Sistema**
1.		El sistema realiza de forma automática la búsqueda de los requisitos y los muestra en la aplicación.
2.	***Prototipo de Interfaz***	



Sección: "Actualizar Requisito"		
	Actor	**Sistema**
1.	Selecciona un requisito en la interfaz principal	El sistema verifica que haya un requisito seleccionado, sino existe ninguno seleccionado ver paso 1 del flujo alterno.
2.	Selecciona la opción "Editar"	Si el requisito es funcional visualiza la interfaz Actualizar Requisito Funcional, sino ver paso 6.
3.	Introduce los datos para el requisito y pulsa actualizar. ✓ Nombre ✓ Estado ✓ Responsable ✓ Prioridad ✓ Complejidad ✓ Dependencias	Valida que los datos introducidos estén correctamente, si son incorrectos ver paso 2 del flujo alterno.
5		Muestra un mensaje de confirmación.

6		Muestra la interfaz Actualizar Requisito No Funcional.
7	Introduce los datos para el requisito y pulsa actualizar. ✓ Nombre ✓ Estado ✓ Responsable ✓ Clasificación ✓ Descripción	Valida que los datos introducidos estén correctamente, si son incorrectos ver paso 2 del flujo alterno.
		Muestra un mensaje de confirmación.
	Prototipo de Interfaz  	

Flujos alternos		
	Actor	**Sistema**
1		Muestra mensaje para indicarle al usuario que debe seleccionar un requisito.
2		Muestra mensaje para indicarle al usuario que no puede actualizar el requisito.

Tabla 3: Descripción textual del Caso de Uso Gestionar Requisito.

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 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. (21)

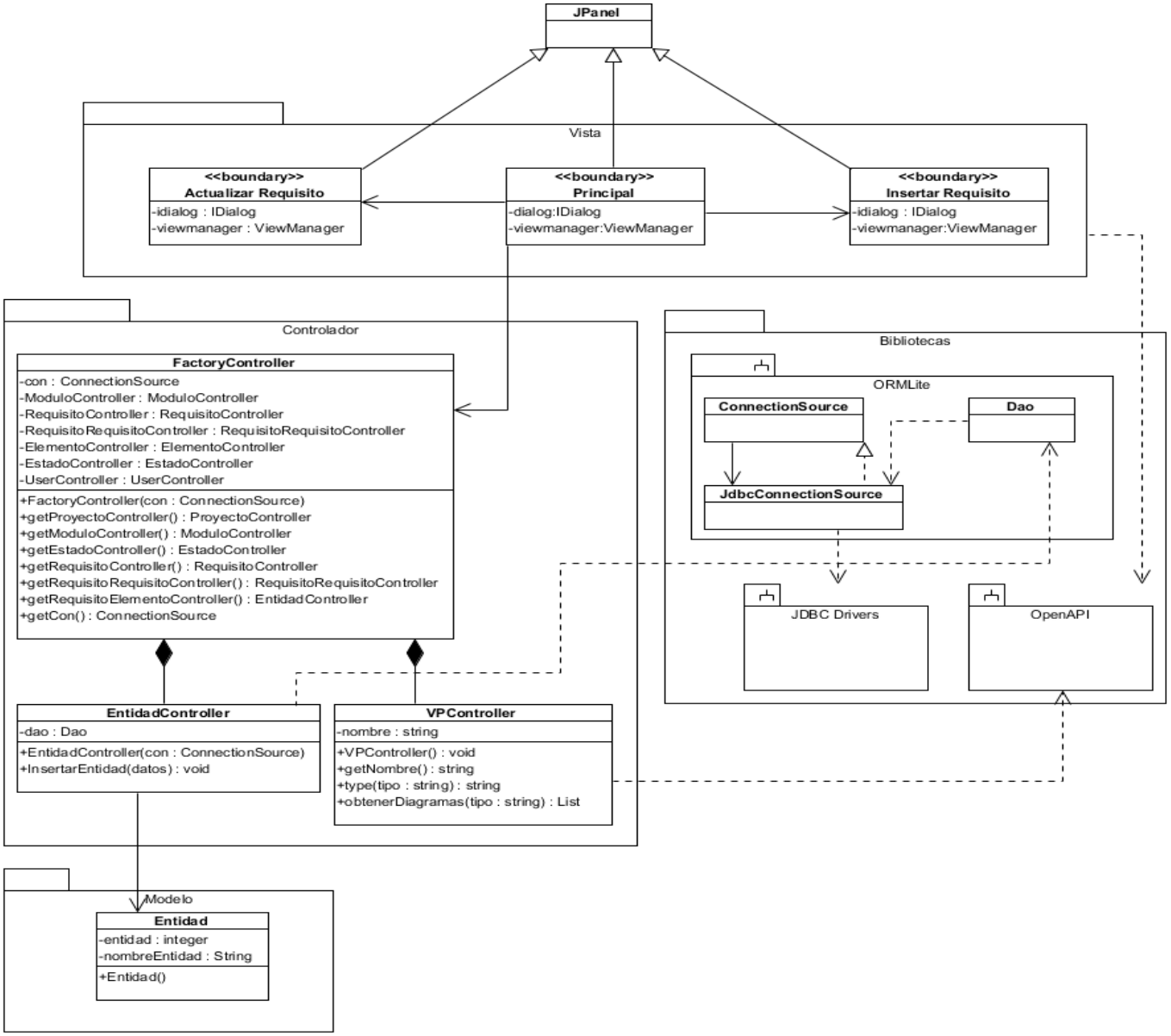


Figura 8: Diagrama de clases del diseño del Caso de Uso Gestionar Requisito.

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

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.

Insertar Requisito: Es la clase interfaz encargada de garantizar la inserción de un requisito.

Actualizar Requisito: Es la clase interfaz encargada de garantizar la actualización de un requisito.

FactoryController: Es la clase encargada de crear todos los controladores de la clases entidades para manejar el acceso a datos.

Bibliotecas: Es el subsistema donde se encuentran las bibliotecas necesarias para realizar la conexión a la base de datos y la biblioteca *OpenAPI* que es el subsistema que permite extender el "*Visual Paradigm for UML"*, la cual es utilizada por las clases interfaces para ser visualizadas en la misma y por la clase *VP_Controller* que es la encargada de realizar las acciones sobre los proyectos cargados en el "*Visual Paradigm for UML"*, gestionando todo lo referente a los diagramas UML que el mismo incluye.

Dao: Es la clase que contiene los objetos de acceso a datos dependiendo de la tabla a la que se requiera acceder, este clase necesita un objeto de la clase *ConnectionSource* para realizar la conexión a la base de datos

ORMLite: Es el subsistema que permite el mapeo de objetos de la base de datos.

ConnectionSource: Es la clase que permite realizar la conexión a la base de datos utilizando la clase **JdbcConnectionSource**.para realizar la conexión mediante los controladores JDBC (Java Database Connectivity por sus siglas en inglés).

JdbcConnectionSource: Esta clase implementa la clase *ConnectionSource*, además de utilizar los controladores de conexión dependiendo del gestor de base de datos para realizar una correcta conexión.

EntidadController: Esta es una representación de todas las clases controladoras que participan en este caso de uso, estas son *ElementoController*, *ModuloController*, *EstadoController*, *RequisitoRequisitoController*, *ProyectoController*, *RequisitoElementoController*, que son las encargadas de realizar las acciones sobre los objetos en las tablas correspondientes.

Entidad: Es una representación de las clases entidades que participan en este caso de uso, estas son *Elemento*, *Modulo*, *Estado*, *RequisitoRequisito*, *Proyecto*, *RequisitoElemento* que contiene los datos de los objetos de dichos nombres.

Patrones de diseño.

Un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico y en medio de "fuerzas" que pueden tener un impacto en la manera en que se aplica y utiliza el patrón. (22)

Patrones GRASP (General Responsibility Assignment Software Patterns) por sus siglas en inglés: Son guías o principios que sirven para asignar responsabilidades a las clases.

Alta Cohesión: Plantea que la información que almacena una clase debe de ser coherente y debe estar, en la medida de lo posible, relacionada con la clase.

Bajo acoplamiento: Plantea que entre las clases debe existir la menor cantidad de dependencias posibles, de tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases.

En la extensión existirán dos tipos de clases, entidades y controladoras, las entidades tendrán los datos de las tablas para el mapeo de la base de datos teniendo una información coherente, pues cada clase entidad será una tabla. La aplicación constará con una clase controladora por cada clase entidad por lo que existirá un bajo acoplamiento, pues las clases controladoras solo dependen de las entidades.

Experto: Este plantea que se debe asignar la responsabilidad a la clase que tiene la información necesaria para cumplirla.

En la extensión las clases controladoras tendrán cada una las funcionalidades para realizar las consultas a la base de datos por ejemplo la clase *RequisitoController* realiza las funciones sobre los requisitos en la tabla correspondiente de la base de datos, y por tanto implementando dicha responsabilidad aplicará el patrón Experto.

Creador: Este guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debe conectar con el objeto producido en cualquier evento. (19)

En la extensión se evidencia este patrón en las clases *FactoryController, RequisitoController, ModuloController, entre otras.* En dichas clases se encuentran implementadas varias acciones dentro de las cuales se crean objetos de otras clases, lo cual evidencia que estas clases son creadoras de las que son instanciadas.

Controlador: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones (controlador) (19):

- ✓ El "sistema" global (controlador de fachada).
- ✓ La empresa u organización global (controlador de fachada).

- ✓ Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- ✓ Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados "Manejador<NombreCasodeUso>" (controlador de casos de uso).

En la extensión existirán diferentes clases controladoras donde se distribuirán las funcionalidades a cada una en específico, por ejemplo, la clase controladora del Usuario (*UsuarioController*) garantizará la inserción, actualización, eliminación y búsqueda de los usuarios en la base de datos, otro ejemplo es la clase *FactoryController* que se encargará específicamente de crear los objetos de las clases controladoras.

Patrones GoF (Gang of Four) por sus siglas en inglés: Dan una solución implementable con su propio diagrama de clases que muestra la forma en que se deben utilizar.

Singleton: Es de tipo creacional, a nivel de objetos. Este garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ésta instancia.

En la extensión se mostrará el uso de este patrón en la clase *VP_REQMAction* que permitirá la creación de una sola instancia de la clase *ViewManager*, manteniendo la consistencia entre los objetos y proporcionando un único punto de acceso a ella.

```
public class VP_REQMAction implements VPActionController {

    private ViewManager viewManager = ApplicationManager.instance().getViewManager();
```

Figura 9: Patrón Singleton en la clase VP_REQMAction.

Iterator: Es de tipo comportamiento a nivel de objetos. Proporciona una forma de acceder secuencialmente a los elementos de un objeto compuesto por agregación sin necesidad de desvelar su representación interna. (23)

Este patrón se verá reflejado en la clase *VPController* poniéndose de manifiesto en el método obtenerCasosUso que permite capturar todos los casos de uso contenidos en el diagrama de casos de uso del sistema modelado y almacenarlos en una lista para su posterior uso.

Patrones arquitectónicos

Definen la estructura general del software, indican las relaciones entre los subsistemas y los componentes del software y definen las reglas para especificar las relaciones entre los elementos (clases, paquetes, componentes, subsistemas) de la arquitectura. (22)

Modelo Vista Controlador

Modelo Vista Controlador o MVC, es un patrón de arquitectura de software que tiene como función dividir una aplicación en tres módulos, el modelo que gestiona los datos según le indique la lógica de control, la vista que maneja la presentación visual de los datos representados por el modelo y el control, que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el modelo.

En la extensión se identificará este patrón, pues la misma estará compuesta por tres paquetes fundamentales, ***VP_REQM.actions.modelo*** donde se encontrarán las clases entidades con la representación de los datos de la aplicación, ***VP_REQM.dialog*** donde se localizarán las vistas de la aplicación que generarán una representación visual del modelo y mostrarán los datos al usuario y por último ***VP_REQM.actions.control*** que contendrá las clases controladoras encargadas de atender la petición del usuario, ejecutando la acción adecuada y creando el modelo pertinente.

Diagramas de secuencia.

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida y con sus interacciones en el tiempo representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Se utilizan para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones, además no están pensados para mostrar lógicas de procedimientos complejos. (24) A continuación se muestra el Diagrama de secuencia del escenario Insertar Requisito del Caso de Uso Gestionar Requisito.

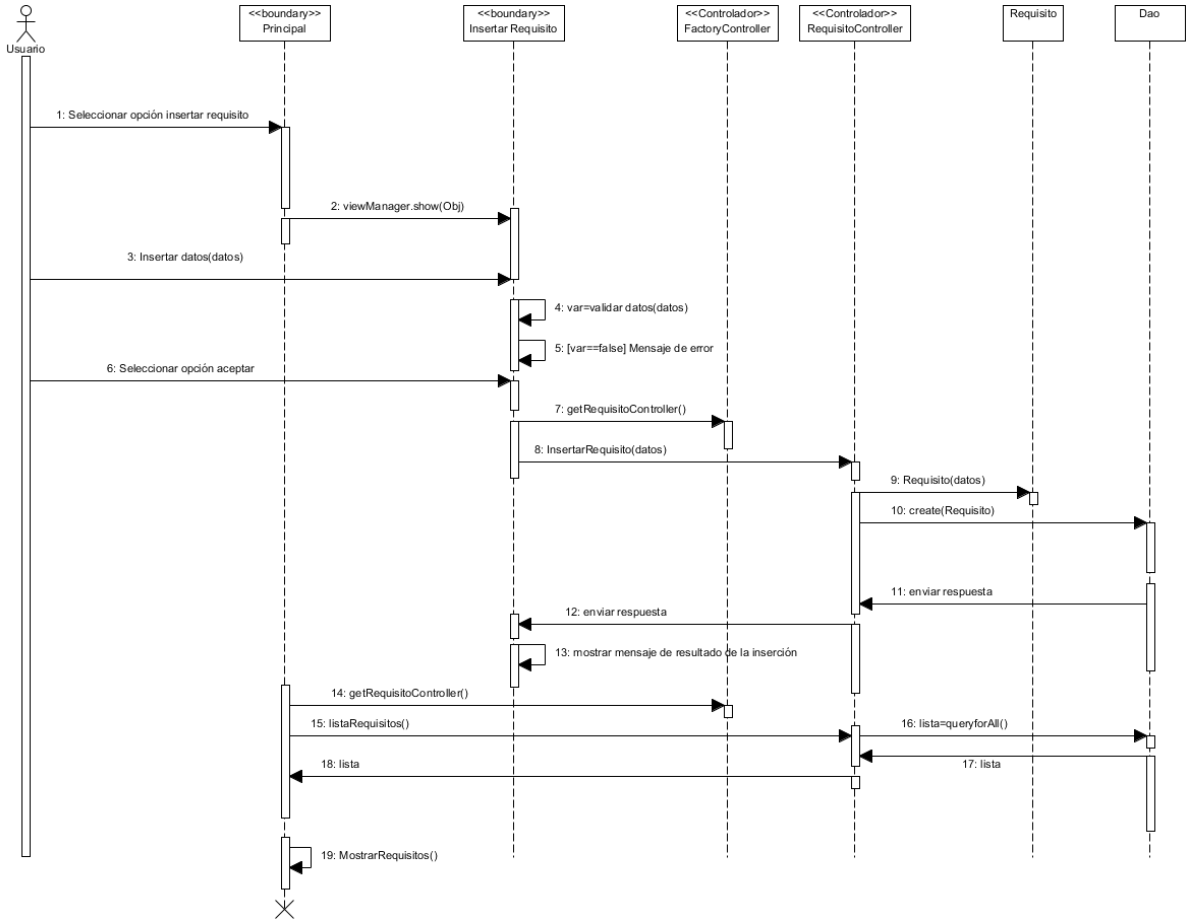


Figura 10: Diagrama de secuencia del escenario Insertar Requisito del CU Gestionar Requisito.

En este diagrama el usuario selecciona la opción Adicionar en la interfaz principal, esta muestra la nueva ventana *Insertar Requisito* para insertar los datos especificados en la descripción del caso de uso. Seguidamente se validan los datos, el usuario selecciona la opción Aceptar y se pide el controlador correspondiente a la clase controladora *FactoryController* para llamar al método InsertarRequisito que se encuentra en el controlador *RequisitoController*, el cual crea una nueva instancia de la clase *Requisito* con los datos del requisito que fueron entrados por el usuario. Después se invoca al método create de la clase *Dao* el que posibilita insertar en la base de datos el nuevo requisito, enviando una respuesta al controlador con el resultado de la inserción, la misma es capturada en la interfaz *Insertar Requisito*, la cual se encarga de manejarla según haya sido el resultado y por último se recargan los datos en la interfaz *Principal*.

Modelo de datos

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. La teoría del modelo de datos relacional es obra del investigador de IBM Edgar Codd en 1970. El modelo relacional se caracteriza a muy grandes rasgos por disponer que toda la información debe estar contenida en tablas, y las relaciones entre datos deben ser representadas explícitamente en esos mismos datos. (25)

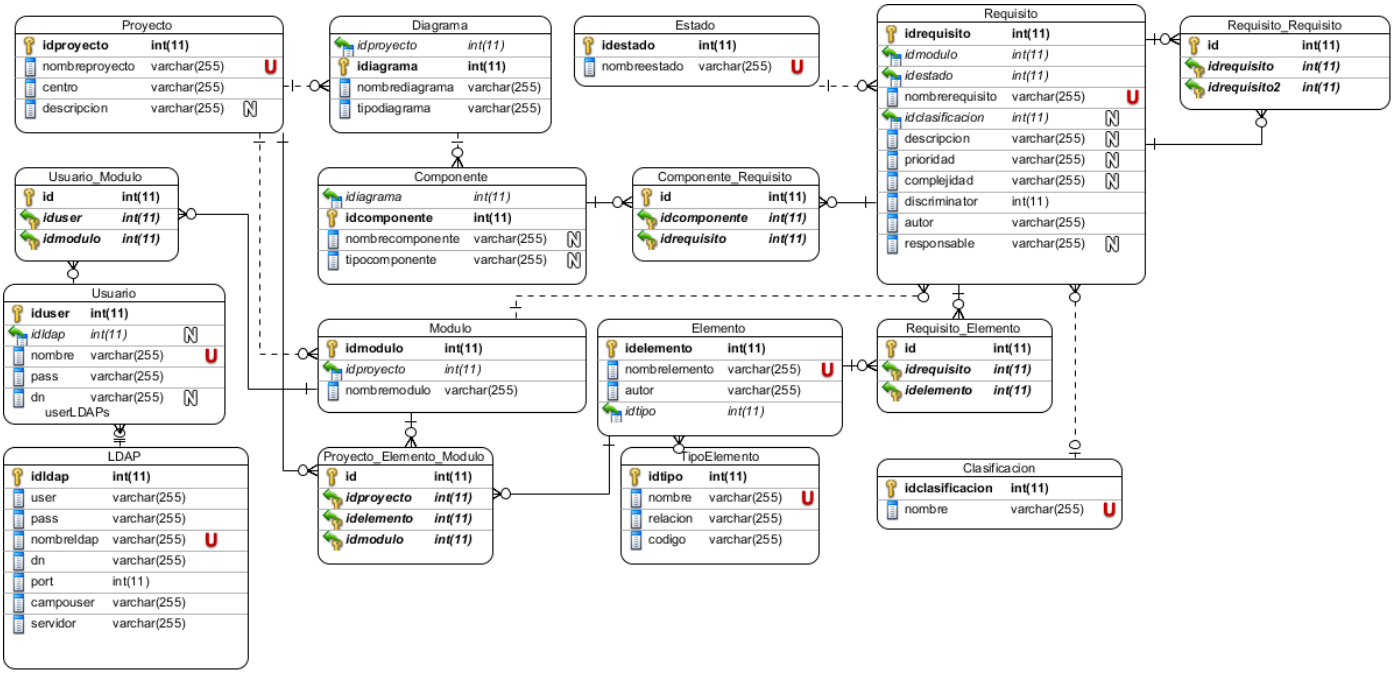


Figura 11: Modelo de datos.

Nombre: Proyecto
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los proyectos insertados.
Relación con la tabla(s): Modulo, Diagrama, Elemento.
Tipo de relación: Uno - Muchos, Uno – Muchos, Muchos - Muchos.
Nombre: Modulo
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los módulos insertados a cada proyecto.
Relación con la tabla(s): Proyecto, Usuario, Requisito, Elemento.

Tipo de relación: Muchos - Uno, Muchos - Muchos, Uno – Muchos, Muchos – Muchos.
Nombre: TipoElemento
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los tipos de **elementos a** tracear.
Relación con la tabla(s): Requisito
Tipo de relación: Muchos – Uno.
Nombre: Elemento
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los elementos a tracear.
Relación con la tabla(s): Requisito, Tipo, Modulo, Proyecto.
Tipo de relación: Muchos – Muchos, Muchos – Uno, Muchos – Muchos, Muchos – Muchos.
Nombre: Estado
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los estados que pueden tener los requisitos insertados.
Relación con la tabla(s): Requisito
Tipo de relación: Uno – Muchos.
Nombre: Requisito
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los requisitos insertados a cada módulo.
Relación con la tabla(s): Modulo, Estado, Elemento, Requisito, Componente.
Tipo de relación: Muchos - Uno, Muchos - Uno, Muchos - Muchos, Muchos – Muchos, Muchos – Muchos.
Nombre: Diagrama
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los diagramas de cada proyecto.
Relación con la tabla(s): Componente, Proyecto
Tipo de relación: Uno – Muchos, Muchos – Uno.
Nombre: LDAP
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los servidores ldap.
Relación con la tabla(s): Usuario

Tipo de relación: Uno - Muchos.
Nombre: Usuario
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los usuarios que utilizarán la aplicación.
Relación con la tabla(s): LDAP, Modulo.
Tipo de relación: Muchos – Uno, Muchos – Muchos.
Nombre: Componente
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a los componentes de cada diagrama.
Relación con la tabla(s): Requisito, Diagrama
Tipo de relación: Muchos – Muchos, Muchos – Uno.
Nombre: Clasificacion
Descripción: Es la tabla en la cual se almacenan los datos pertenecientes a las clasificaciones de los requisitos.
Relación con la tabla(s): Requisito
Tipo de relación: Uno – Muchos.

Tabla 4: Descripción de las tablas del modelo de datos.

Las tablas *Usuario_Modulo, Requisito_Elemento, Componente_Requisito y Requisito_Requisito* son tablas generadas a partir de relaciones de muchos a muchos, esta última garantiza la relación existente entre requisitos en caso de existir dependencias entre los mismos. La tabla *Proyecto_Elemento_Modulo*, es una relación ternaria entre las tablas que conforman la misma posibilitando que un elemento se encuentre en un mismo proyecto en diferentes módulos.

Modelo de despliegue

El Modelo Físico o de Despliegue provee un modelo detallado de la forma en la que los componentes se desplegarán a lo largo de la infraestructura del sistema. Detalla las capacidades de red, las especificaciones del servidor, los requisitos de hardware y otra información relacionada al despliegue del sistema propuesto. (26)

El Diagrama de Despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware y muestra cómo los elementos y artefactos del software se trazan en nodos.

- ✓ **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.

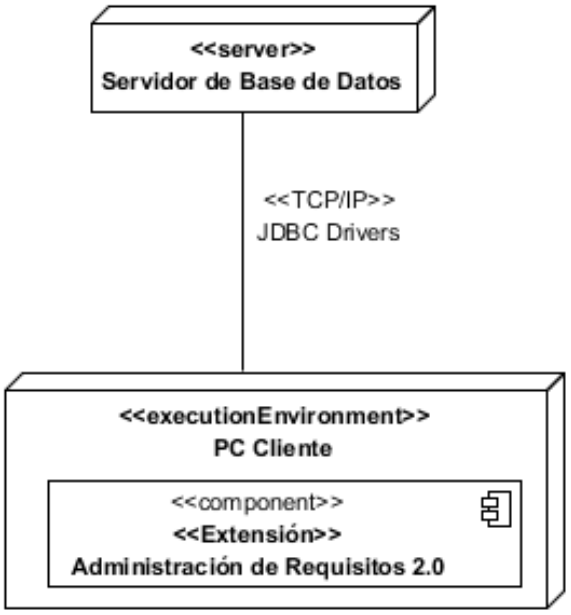


Figura 12: Diagrama de Despliegue.

En este diagrama se identifican dos nodos principales: la PC Cliente que es donde estará disponible la extensión y el servidor de base de datos en el que se encontrará instalado el gestor de bases de datos. La PC Cliente y el servidor de base de datos se comunicarán a través del protocolo TCP/IP utilizando los controladores JDBC dependiendo del gestor de base de datos.

2.5 Conclusiones parciales del capítulo

En este capítulo como parte del análisis y diseño de la extensión se realizó una descripción de la misma con el fin de delimitar aquello que debe ser refinado e implementado para así solucionar la problemática identificada, en este sentido se definieron los siguientes aspectos:

- ✓ Se modeló el diagrama de clases conceptuales con el objetivo de lograr un entendimiento para el desarrollo de la extensión.
- ✓ Se identificaron los requisitos que soportará el desarrollo de la extensión, desglosados en 50 requisitos funcionales y 8 requisitos no funcionales, agrupados en las categorías de seguridad, software, hardware y restricciones del diseño e implementación.
- ✓ Como parte de la vista externa del sistema se confeccionó, aplicando patrones de casos de uso, el diagrama de casos de uso del sistema evidenciándose la relación entre actores y casos de uso.
- ✓ La vista estática del sistema se garantizó a través de los diagramas de clases del diseño como parte de la realización de los casos de uso del sistema, los cuales posibilitaron realizar la descripción de la estructura de la extensión mostrando las clases, atributos, interfaces y las relaciones entre ellos, así como la aplicación de los patrones de diseño que posibilitaron resolver problemas en el diseño de la extensión.
- ✓ El patrón arquitectónico utilizado, Modelo Vista Controlador, permitió la definición de la estructura general de la extensión y propició una mayor organización al dividirla en tres módulos.
- ✓ Los diagramas de secuencia elaborados permitieron describir gráficamente cada escenario de cada caso de uso, así como una vista dinámica de la extensión. El modelo de datos relacional especificó las clases persistentes de la base de datos y finalmente con la confección del diagrama de despliegue se logró modelar la distribución física del sistema.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA DE LA EXTENSIÓN.

En el presente capítulo se realiza el diagrama de componentes de la extensión, se describe la organización del código fuente, así como los estándares de codificación del mismo. Además se muestran los diferentes tipos de pruebas, las cuales tributan al correcto funcionamiento de la propuesta de software presentada en esta investigación, logrando así que se cumpla con las especificaciones propuestas por el cliente.

3.1 Modelo de implementación.

El modelo de implementación es como un conjunto de componentes y subsistemas de implementación que constituyen la composición física de la implementación del sistema. Entre los componentes se pueden encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. Este artefacto describe cómo se implementan los componentes, congregándolos en subsistemas organizados en capas y jerarquías, y señala las dependencias entre éstos. Para representar los diagramas del Modelo de Implementación se puede emplear el diagrama de Componentes. (27)

Diagrama de Componentes

El diagrama de componentes muestra como el sistema está dividido en componentes y las dependencias entre ellos. 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. Este diagrama provee una vista arquitectónica de alto nivel del sistema, ayuda a los desarrolladores a visualizar el camino de la implementación, permitiendo tomar decisiones respecto a las tareas de implementación. (28)

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.

A continuación se representa el diagrama de componentes referente al caso de uso ***Gestionar Requisito***, el cual describe cada uno de los componentes asociados al diseño de clases propuesto para el caso de uso, así como la relación de dependencia entre los componentes que integran la extensión.

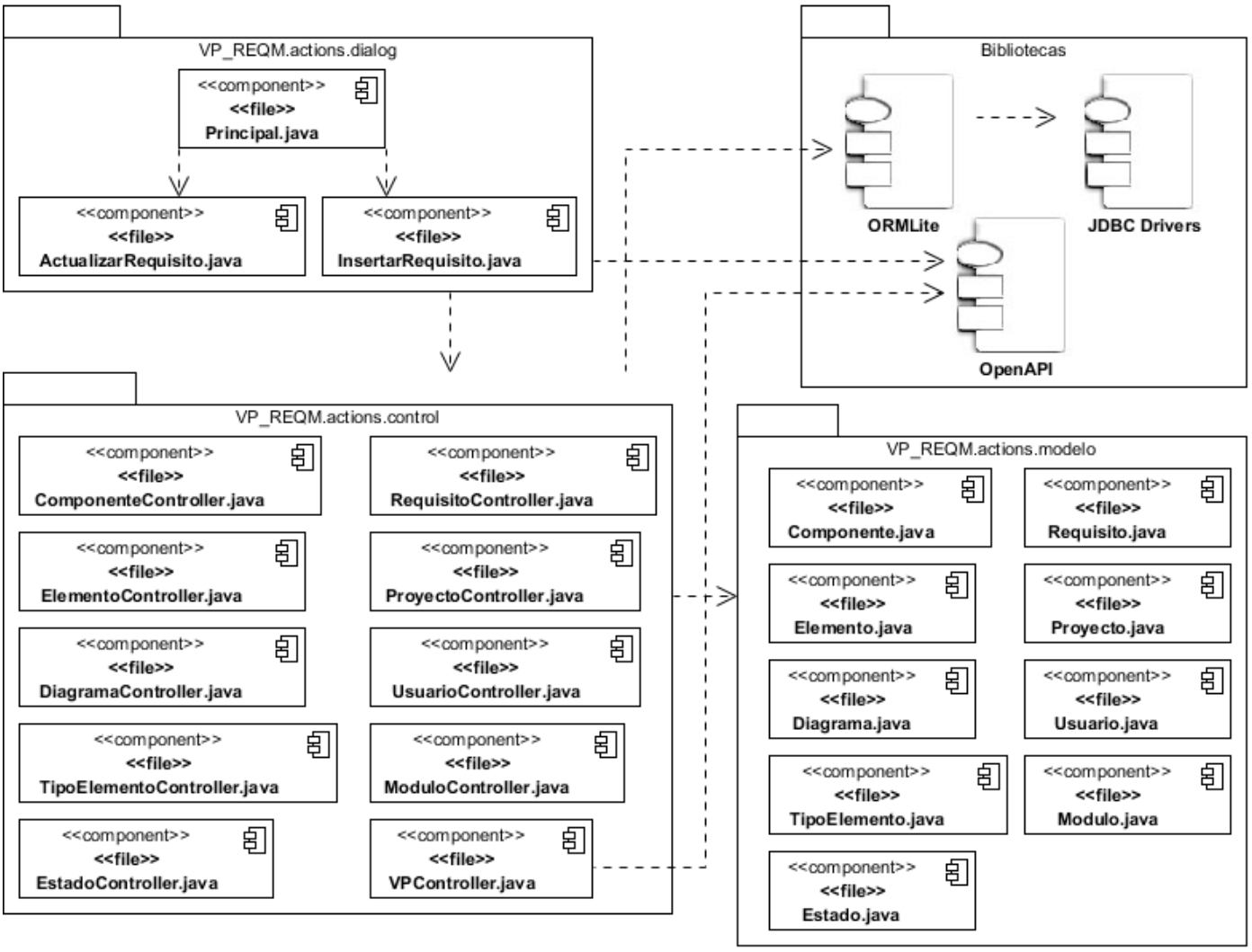


Figura 13: Diagrama de Componentes Caso de Uso Gestionar Requisito.

En el diagrama se puede identificar cuatro paquetes fundamentales que componen a la extensión, ***VP_REQM.actions.dialog*** que es el paquete donde se encuentran las interfaces que utilizan las clases controladoras para atender las peticiones de los usuarios, por tanto se relaciona con ***VP_REQM.actions.control***, pues es donde se encuentran las controladoras que actúan sobre los datos representados por el modelo, por lo que se relaciona con ***VP_REQM.actions.modelo*** que es en el cual están las clases entidades utilizadas por las controladoras que tienen los atributos necesarios para el mapeo de la base de datos.

Bibliotecas: Es donde se encuentran las bibliotecas necesarias para realizar la conexión a la base de datos y la biblioteca *OpenAPI* que permite extender el "*Visual Paradigm for UML"*.

3.2 Código fuente

El **código fuente**, o ***source code***, también llamado código base, es un texto que se ha escrito en un lenguaje de programación concreto, y que sólo puede ser leído por un experto o programador. Es un programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de máquina mediante compiladores, ensambladores o intérpretes. (29)

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, este 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, debe establecerse 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. (30)

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. (30)

Para el desarrollo de la extensión se definió un estándar de codificación el cual se explica a continuación.

Número de declaraciones por línea: Se debe declarar cada variable en una línea distinta, de esta forma cada variable se puede comentar por separado. Ejemplo:

```
private IDialog dialog;
private ViewManager manager;
private FactoryController factory;
private Usuario user;
private ConnectionSource con;
private DefaultTableModel modelElem;
private DefaultTableModel modelReq;
```

Localización: En el comienzo de los bloques no se deben declarar variables con el mismo nombre que otras en un método. La única excepción es el ciclo ***for*** y el ***while***.

Espacio en blanco: Se debe usar una línea en blanco entre:

- ✓ Métodos.
- ✓ Variables locales de un método y la primera sentencia.
- ✓ Entre diferentes secciones lógicas dentro de un fichero (más legibilidad).

Ejemplo:

```
public void activarElementoInicio() {
    int elem = ElementosExistentes.getModel().getRowCount();

    addelem.setEnabled(true);
    if (elem != 0) {
        panelementos.setEnabled(true);
        editelem.setEnabled(true);
        deletelem.setEnabled(true);
    } else {
        panelementos.setEnabled(true);
        editelem.setEnabled(false);
        deletelem.setEnabled(false);
    }
}
```

Asignación de nombres: Cada tipo de elemento debe nombrarse con una serie de reglas determinadas.

Clases e interfaces: Nombres. La inicial en mayúscula. Ejemplo:

```
 * @author jorge
 */
public final class VistaAnalista extends javax.swing.JPanel implements IDialogHandler {
```

Métodos: Deben ser verbos. La primera letra de la primera palabra en minúsculas, el resto de las palabras empiezan por mayúsculas. Ejemplo:

```
public void actualizarContraseña(Usuario user,String pass) throws SQLException {
    user.setPass(pass);
    dao.update(user);
}
```

Variables: Deben comenzar por minúscula. No se utilizará en ningún caso el carácter "_". Ejemplo:

```
private Usuario user;
```

3.3 Pruebas de software a la extensión

Durante el proceso de desarrollo de software se hace necesario realizar pruebas con el objetivo de detectar fallos en la aplicación, dicho proceso permite mantener la calidad del producto y verifica que se cumplan los requisitos iniciales.

Como parte del proceso se realizaron los siguientes tipos de prueba:

Pruebas Unitarias

Rod Johnson define las pruebas unitarias como *"el nivel más fino de granularidad en las pruebas, que verifican una simple unidad de funcionalidad y deben probar que cada método de una clase satisface su contrato documentado".* (31)

Al desarrollar un nuevo software, la primera etapa de pruebas a considerar es la etapa de pruebas unitarias. Es la prueba enfocada a los elementos testeables más pequeños del software, estas aseguran que un único componente de la aplicación produzca una salida correcta para una determinada entrada. Es por esto que las pruebas unitarias se basan en el método de caja blanca.

Método de caja blanca

La prueba de caja blanca es un método utilizado para el diseño de casos de pruebas que se basan en un análisis detallado de la estructura del código, se comprueban los caminos lógicos, bucles y condiciones examinando así el estado del programa para informar de la situación real de la calidad del software. (32)

Para realizar las pruebas unitarias a la extensión mediante el método de caja blanca se utilizó la biblioteca JUnit, a continuación se muestra un ejemplo de los resultados obtenidos al aplicarse las pruebas a las funcionalidades de la clase *GenerarMatriz*, realizadas en tres iteraciones:

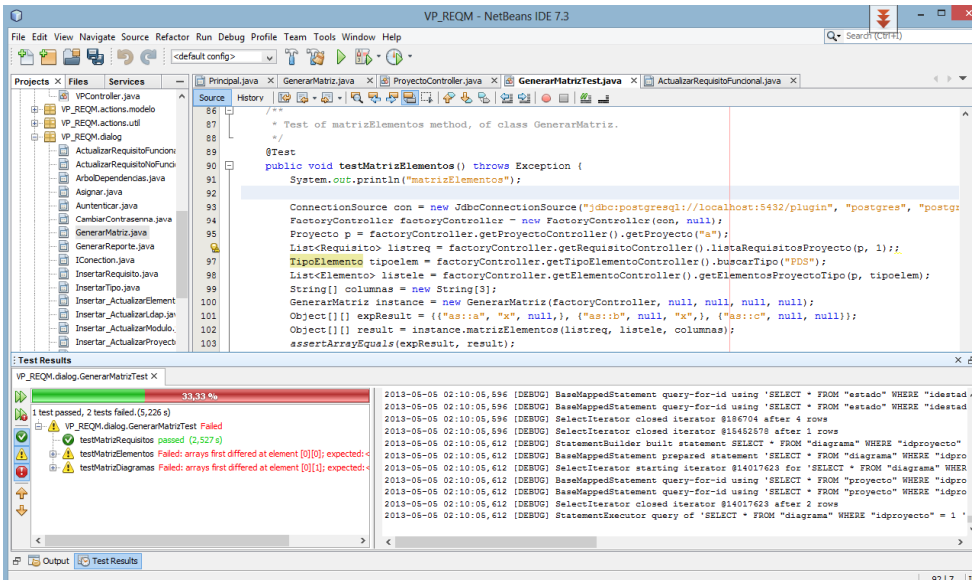


Figura 14: Prueba Unitaria - Primera iteración.

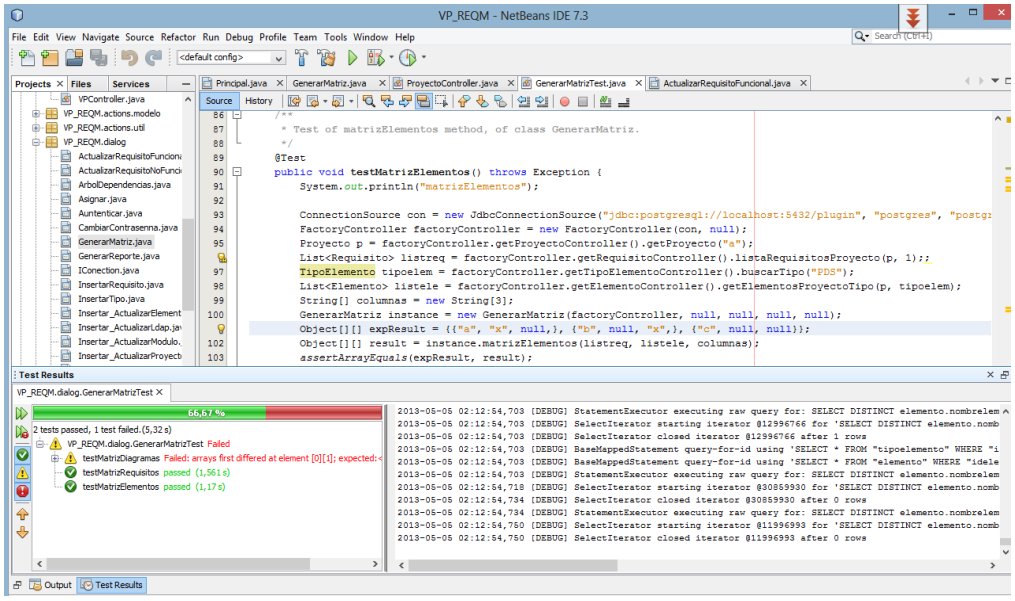


Figura 15: Prueba Unitaria - Segunda iteración.

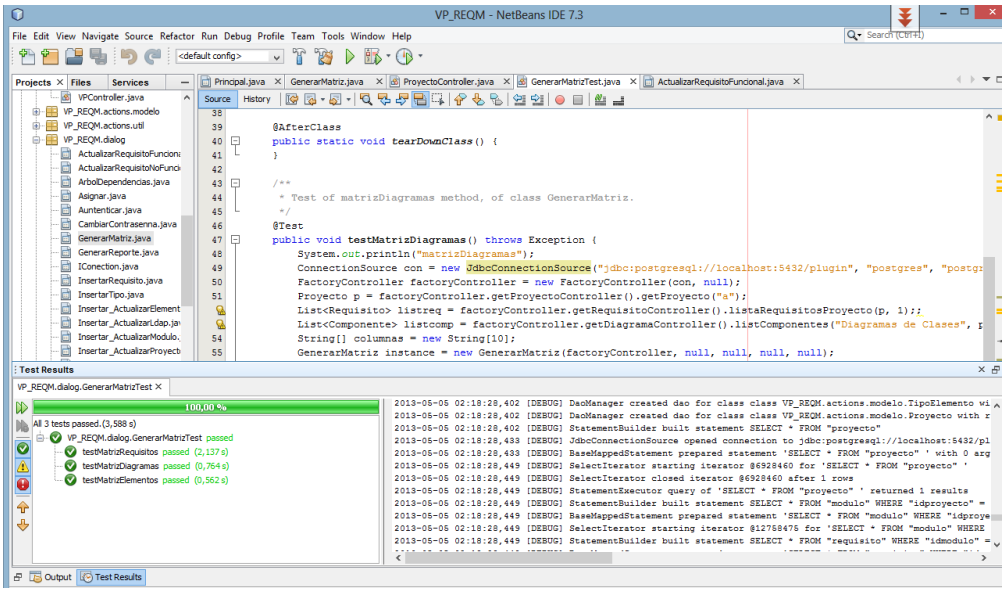


Figura 16: Prueba Unitaria - Tercera iteración.

Como producto de todo el proceso de prueba de caja blanca se obtuvo un progreso en cada iteración. Se detectaron errores en la primera iteración observándose un cumplimiento de un 33,33%, posteriormente fueron suprimidos algunos de estos errores adquiriendo un 66,67% en la segunda iteración y finalmente en una tercera iteración se logró un resultado satisfactorio en todas las funcionalidades.

Pruebas de Integración

Según Roger Pressman *"Las pruebas de integración son una técnica sistemática para la construcción de la arquitectura de software, mientras al mismo tiempo, se aplican las pruebas para descubrir los errores asociados con la interfaz".*

Existen dos tipos de integración, no incremental e incremental. No incremental es cuando se combinan todos los módulos y se prueba todo el programa en su conjunto y la integración incremental es cuando el programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y corregir. (33) Existen dos estrategias de integración incremental: Incremental ascendente e Incremental descendente. Para realizar la prueba de integración a la extensión se aplicó la estrategia de integración ascendente, la cual consiste en comenzar con los módulos de nivel inferior, y verificar que los módulos de nivel inferior llamen a los de nivel superior de manera correcta, con los parámetros correctos.

La herramienta *"Visual Paradigm for UML"* permite ser extendida mediante la biblioteca openapi.jar, pero antes de realizar la integración con el *"Visual Paradigm for UML"* hay que tener en cuenta varias

condiciones para realizar la extensión que son vitales para la misma. Primeramente se debe respetar la estructura que sigue la extensión para su implementación.

La misma presenta un paquete general con tres paquetes asociados, el primero con el nombre de la extensión el cual representa las configuraciones de la misma, el segundo paquete contiene las acciones definidas para realizar y el último paquete presenta los diálogos o formularios. Además de estos se incluyeron dentro del paquete de acciones para garantizar un orden lógico en la implementación tres paquetes, los cuales son: *modelo,* el cual contiene las clases entidades de la base de datos, *control* que contiene las clases controladoras de la aplicación y *util* que contiene un conjunto de clases para redefinir propiedades de algunos componentes, encriptar contraseña y para realizar la conexión LDAP, entre otras.

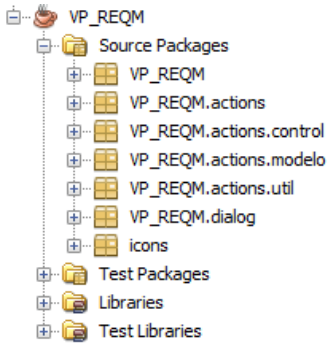


Figura 17: Estructura general de paquetes.

Los ficheros más importantes para la integración son *plugin.xml*, *VP_REQM.java* que permiten cargar y configurar la extensión, además del *VP_REQMAction.java* que define el performAction el cual permite ejecutar acciones referentes al evento onClick de la acción.

EL segundo paso es que todos los formularios hereden de la clase *JPanel* e implementen la interfaz *IDialogHandler*, la primera es la que permite tener los componentes en el "*Visual Paradigm for UML*" y la segunda permite visualizarlos en el mismo.

```
public class Principal extends javax.swing.JPanel implements IDialogHandler {
```

Figura 18: Código del formulario.

El tercer 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.

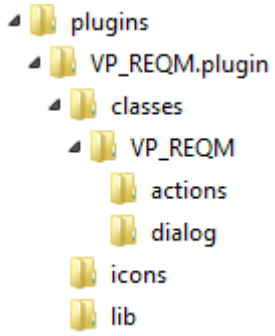


Figura 19: Estructura de despliegue.

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. Para esto se realizó la implementación de una nueva herramienta que facilitará el proceso.

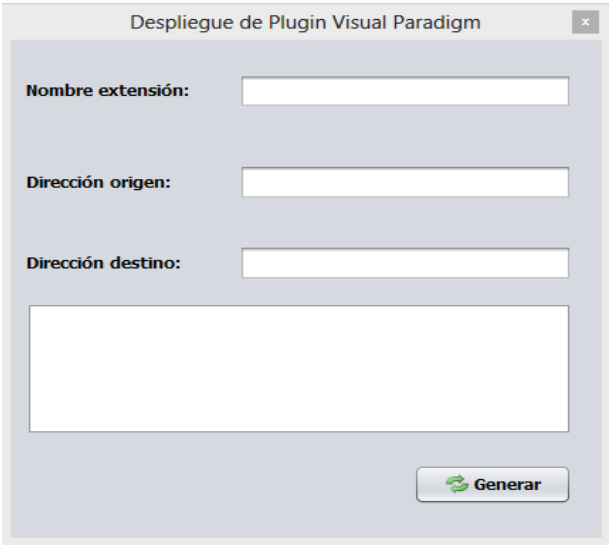


Figura 20: Herramienta para el despliegue de la extensión.

Pruebas funcionales a nivel de Sistema.

Según Ian Sommerville *"Las pruebas del sistema consisten en la integración de dos o más componentes que implementan las funciones o características del sistema y las pruebas de integración al mismo".*

Estas se realizan para verificar que todos los módulos trabajan como sistema sin error. Las pruebas del sistema examinan qué tan bien el sistema cumple con los requisitos de la organización y su utilidad, seguridad y desempeño. Entre las técnicas de prueba que se realizan en un sistema se encuentran las pruebas de funcionalidad las cuales como su nombre lo indica se encargan de evaluar la funcionalidad del sistema. Dentro de las pruebas de funcionalidad se realizan las pruebas funcionales, las que tienen como objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. Este tipo de prueba se basa en el método de caja negra.

Método de caja negra.

Las pruebas de caja negra no consideran codificación dentro de sus parámetros a evaluar, es decir, que no están basadas en el conocimiento del diseño interno del programa. Estas pruebas se enfocan en los requerimientos establecidos y en la funcionalidad del sistema.

Se ejecuta cada caso de uso, flujo de caso de uso, o función, usando datos válidos e inválidos, para verificar lo siguiente (22):

- ✓ Que se aplique apropiadamente cada regla de negocio.
- ✓ Que los resultados esperados ocurran cuando se usen datos válidos.
- ✓ Que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.

En el diseño de casos de prueba se utilizó la técnica de partición de equivalencia que es una técnica del método de prueba de caja negra que divide el campo de entrada de un programa en variables con juegos de datos de entrada y salida. En esencia, esta técnica intenta dividir el dominio de entrada de un programa en un número finito de variables de equivalencia. De tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada variable es equivalente a una prueba realizada con cualquier otro valor de dicha variable.

Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Éstas se identifican examinando cada condición de entrada

(normalmente una frase en la especificación) y dividiéndola en dos o más grupos. Se definen dos tipos de variables de equivalencia, las válidas, que representan entradas válidas al programa, y las no válidas, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor. (34)

Casos de Prueba.

Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para conseguir un objetivo particular o condición de prueba como, por ejemplo, verificar el cumplimiento de un requisito específico. (34) A continuación se presenta la matriz de datos para el caso de uso Gestionar Requisito, al cual se le realizan las pruebas mediante la misma para comprobar que funcione de forma correcta, donde:

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	*servidor*	*campo de texto*	*no*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*
2	*puerto*	*campo de texto*	*no*	*Alfanumérico de 1 a 255 -.*
3	*nombre base de datos*	*campo de texto*	*no*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*
4	*usuario*	*campo de texto*	*no*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*
5	*contraseña*	*campo de texto password*	*no*	*Alfanumérico de 1 a 255 caracteres. Admite 0-9 a-z A-Z - áéíóú ñÑ ÓÁÍÉÚ üÜ _ ( ) , ; . : / ^ - [ ] {}% $ # @ * " &*

Tabla 5: Variables de entrada para el Caso de Prueba del CU Establecer Conexión.

Escenario	Descripción	1	2	3	4	5	Respuesta del sistema	Flujo central
EC 1.1 Insertar datos de conexión. Correctamente	Se adicionan correctamente los datos.	V (10.8.97.222)	V (5432)	V (plugin)	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 conectar
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 (plugin)	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 conectar
		V (10.8.97.222)	N/A	V (plugin)	V (postgres)	V (Postgres)		
		V (10.8.97.222)	V (5432)	N/A	V (postgres)	V (Postgres)		
		V (10.8.97.222)	V (5432)	V (plugin)	N/A	V (Postgres)		
		V (10.8.97.222)	V (5432)	V (plugin)	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 (plugin)	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 conectar
		V (10.8.97.222)	I (puerto incorrecto)	V (plugin)	V (postgres)	V (Postgres)		
		V (10.8.97.222)	V (5432)	I (nombre base de datos	V (postgres)	V (Postgres)		
		V (10.8.97.222)	V (5432)	V (plugin)	I (usuario no existente)	V (Postgres)		
		V (10.8.97.222)	V (5432)	V (plugin)	V (postgres)	I (contraseña incorrecta)		

Tabla 6: Caso de Prueba del CU Establecer Conexión con la Base de Datos.

Después de realizar las pruebas de Caja Negra a todos los casos de prueba de la extensión los cuales fueron realizados para cada caso de uso, 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. A continuación se muestran los resultados de las pruebas realizadas al CU Establecer Conexión con la Base de Datos y el total a toda la extensión, las cuales arrojaron los siguientes resultados.

Diseño de caso de prueba	No conformidades pendientes		
	Iteración 1	Iteración 2	Iteración 3
Establecer Conexión con la Base de Datos	6	2	0
Total	**37**	**10**	**0**

Tabla 7: Resultados de las pruebas realizadas a la extensión.

.

Para ver todos los Casos de Prueba referentes a los casos de uso ver (Expediente de Proyecto).

3.4 Conclusiones parciales del capítulo

En este capítulo como parte de la implementación y pruebas de la extensión se realizaron los siguientes pasos:

- ✓ Se estructuró el modelo de implementación a través de los diagramas de componentes, los cuales permitieron mostrar los componentes y sus relaciones sobre la base de la arquitectura definida para el desarrollo de la extensión.
- ✓ Se realizó la implementación de la extensión aplicando los estándares de codificación definidos.
- ✓ Como parte de la validación de la extensión y su correcto funcionamiento se realizaron las pruebas unitarias a nivel de desarrollador utilizando la biblioteca JUnit, pertenecientes al método de caja blanca; las pruebas funcionales a nivel de integración y por último las pruebas funcionales a nivel de sistema basada en la técnica de partición de equivalencia del método de caja negra, guiados por los diseños de casos de prueba basados en casos de uso.
- ✓ Se elaboró una guía de instalación y configuración de la extensión y se le incorporó una ayuda a la misma para que los usuarios finales tengan un mejor entendimiento de la herramienta.

CONCLUSIONES

Una vez concluido el desarrollo de la solución propuesta se arribó a las siguientes conclusiones:

- ✓ Se identificaron los requisitos que delimitaron el desarrollo de la extensión, desglosados en 50 requisitos funcionales y 8 requisitos no funcionales.
- ✓ Se realizó el diseño de la extensión a través de los diagramas de clases y de secuencia, aplicando patrones de diseño y sobre la base del patrón arquitectónico MVC.
- ✓ Se implementaron las funcionalidades identificadas, se validó y comprobó el correcto funcionamiento de la extensión a través de las pruebas unitarias, de integración y sistema.

RECOMENDACIONES

- ✓ Actualizar los diagramas cargados en el "*Visual Paradigm for UML*" a través de la extensión para optimizar la gestión de los requisitos y la trazabilidad.

REFERENCIAS

1. **INTECO.** *Guía Avanzada de Gestión de Requisitos.* 2008.

2. **Córdava, Carlos.** *Implementación de requsitos no funcionales a través de la programación orientada a aspectos.* 2007.

3. **Grupo Comex.** [En línea] [Citado el: 15 de Noviembre de 2012.] http://www.grupocomex.com/REQM-Gestion-de-Requisitos.aspx.

4. **IBM.** IBM. [En línea] [Citado el: 16 de Noviembre de 2012.] http://www-142.ibm.com/software/products/es/es/reqpro.

5. **Sánchez Pescador, Elena.** *Análisis e implantación de una herramienta de gestión de requisitos para la gestión de servicios basado en la filosofía de ITL V3, CMMI de servicios y MOF.* 2009.

6. **IpCorp.** IpCorp. [En línea] [Citado el: 16 de Noviembre de 2012.] http://www.ipcorp.com.ar/blog/?s=OSRMT.

7. **Sparx systems.** [En línea] [Citado el: 21 de Noviembre de 2012.] http://www.sparxsystems.com.ar/products/ea.html.

8. **INTECO.** *Ingeniería del Software: Metodologías y ciclo de vida.*

9. **eclipse.** [En línea] [Citado el: 23 de Noviembre de 2012.] http://www.eclipse.org/epf/general/OpenUP.pdf.

10. **Hernández Orallo, Enrique. disca.** [En línea] http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

11. **Kioskea.** [En línea] [Citado el: 23 de Noviembre de 2012.] http://es.kioskea.net/contents/langages/langages.php3.

12. **Zukowski, John.** *Java 2. J2SE 1.4.* La Habana : Félix Varela, 2007.

13. **Scribd.** [En línea] [Citado el: 25 de Noviembre de 2012.] http://es.scribd.com/doc/94927017/Ides.

14. **Netbeans. Netbeans.** [En línea] [Citado el: 4 de Diciembre de 2012.] http://www.netbeans.org/index_es.html.

15. **Slideshare.** [En línea] [Citado el: 7 de Diciembre de 2012.] http://www.slideshare.net/guestf131a9/herramientas-case.

16. **Visual Paradigm. Visual Paradigm .** [En línea] [Citado el: 10 de Diciembre de 2012.] http://www.visual-paradigm.com/product/vpuml/.

17. **Python.** [En línea] [Citado el: 10 de Diciembre de 2012.] http://www. python.org.ar/pyar/ORMs.

18. **ormlite.** [En línea] [Citado el: 10 de Diciembre de 2012.] http://ormlite.com.

19. **Larman, Craig.** *UML y patrones 2da Edición.* Prentice Hall : s.n., 2003.

20. **Sparxsystems.** [En línea] [Citado el: 18 de Febrero de 2013.] http://www.sparxsystems.com.ar/uml-tutorial.html.

21. **Booch, y otros.** *The Unified Modeling Language User Guide.* Massachusetts : Addison-Wesley Longman Inc, 1998. 0-201-57168-4**.**

22. **Pressman, Roger S.** *Ingeniería del Software Un enfoque práctico 6ta Edición.* La Habana : Félix Varela, 2005.

23. **upm.** [En línea] [Citado el: 18 de Febrero de 2013.] http://is.ls.fi.upm.es/docencia/proyecto/docs/patrones_gof.pdf.

24. **Sparxsystems.** [En línea] [Citado el: 19 de Febrero de 2013.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.html.

25. **umad.** [En línea] [Citado el: 21 de Febrero de 2013.] http://umad.zapatolibre.com/moodle_files/Articulo - El modelo relacional de bases de datos - Codd.pdf.

26. **Sparxsystems.** [En línea] [Citado el: 25 de Marzo de 2013.] http://www.sparxsystems.com.ar/resources/tutorial/physical_models.html.

27. **merinde.** [En línea] [Citado el: 20 de Marzo de 2013.] http://merinde.net/index.php?option=com_content&task=view&id=495&Itemid=291.

28. **msdn.microsoft.** [En línea] [Citado el: 23 de Marzo de 2013.] http://msdn.microsoft.com/es-es/library/dd409390.aspx.

29. **onsoni.** [En línea] [Citado el: 24 de Marzo de 2013.] http://www.onsoni.com/2011/11/definicion-del-codigo-fuente.html.

30. **serk.kualtus.** [En línea] [Citado el: 25 de Marzo de 2013.] http://serk.kualtus.com/codigo.htm.

31. **R , JOHNSON y J, HOELLER.** *Expert One-on-One™ J2EE™ Development without EJB™.* (2004.).

32. **Fuentes Guerra, Yurién Ricardo y Jordán Borjas, Ernesto.** *Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca.* Ciudad de La Habana : s.n., 2008.

33. **Lores Sánchez, Linet y Monné Roque , Diana .** *Aplicación de las pruebas de liberación al Sistema Informático de Genética Médica.* Ciudad de La Habana : s.n., 2009.

34. **cibertec.** [En línea] [Citado el: 25 de Abril de 2013.] http://cibertec.googlecode.com/files/Pruebas de Software.pdf.

BIBLIOGRAFÍA

1. **Alarcón, Andrea y SANDOVAL, Erika.** *Herramientas CASE para ingeniería de Requisitos.* Cataluña : s.n., 2008.

2**. Booch, y otros.** *The Unified Modeling Language User Guide.* Massachusetts : Addison-Wesley Longman Inc, 1998. 0-201-57168-4.

3. **cibertec.** [En línea] [Citado el: 25 de Abril de 2013.] http://cibertec.googlecode.com/files/Pruebas de Software.pdf.

4. Córdava, Carlos. *Implementación de requsitos no funcionales a través de la programación orientada a aspectos.* 2007.

5. Durán Toro, Amador y Bernárdez Jiménez, Beatriz. *Metodología para la Elicitación de Requisitos de Sistemas Software.* Sevilla : s.n., 2000.

6. eclipse. [En línea] [Citado el: 23 de Noviembre de 2012.] http://www.eclipse.org/epf/general/OpenUP.pdf.

7. eva. [En línea] [Citado el: 21 de Noviembre de 2012.] http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodologias/Evolucion_de_Metodologias.pdf.

8. eva. [En línea] [Citado el: 21 de Noviembre de 2012.] http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodologias/METODOLOGIAS_TRADICIONALES_VS._METODOLOGIAS_AGILES.pdf.

9**. Fuentes Guerra, Yurién Ricardo y Jordán Borjas, Ernesto.** *Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca.* Ciudad de La Habana : s.n., 2008.

10. **Gamma, Erich, y otros.** *Design Patterns Elements of Reusable Object-Oriented Software.* 1994.

11. **Grupo Comex.** [En línea] [Citado el: 15 de Noviembre de 2012.] http://www.grupocomex.com/REQM-Gestion-de-Requisitos.aspx.

12. **Gunnar Övergaard, Karin Palmkvist.** *Use Cases Patterns and Blueprints.* s.l. : Addison Wesley Professional, 2004. 0-13-145134-0.

13. **Hernández Orallo, Enrique. disca.** [En línea] http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

14. **Holzner, Steve.** *Design Patterns For Dummies.* 2006.

15. **IBM.** IBM. [En línea] [Citado el: 16 de Noviembre de 2012.] http://www-142.ibm.com/software/products/es/es/reqpro.

16. **INTECO.** *Guía Avanzada de Gestión de Requisitos.* 2008.

17. **INTECO.** *Ingeniería del Software: Metodologías y ciclo de vida.*

18**.IpCorp.** IpCorp. [En línea] [Citado el: 16 de Noviembre de 2012.] http://www.ipcorp.com.ar/blog/?s=OSRMT.

19. **It news***.* [En línea] [Citado el: 21 de February de 2013.] www.itnews.ec/marco/000155.aspx.

20. **Kioskea.** [En línea] [Citado el: 23 de Noviembre de 2012.] 21. http://es.kioskea.net/contents/langages/langages.php3**.**

22. **lab.inf.uc3m.** [En línea] [Citado el: 13 de Mayo de 2013.] http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html.

23. **Larman, Craig.** *UML y patrones 2da Edición.* Prentice Hall : s.n., 2003.

24. **Lores Sánchez, Linet y Monné Roque , Diana .** *Aplicación de las pruebas de liberación al Sistema Informático de Genética Médica.* Ciudad de La Habana : s.n., 2009.

25. **merinde.** [En línea] [Citado el: 20 de Marzo de 2013.] http://merinde.net/index.php?option=com_content&task=view&id=495&Itemid=291.

26. **msdn.microsoft.** [En línea] [Citado el: 23 de Marzo de 2013.] http://msdn.microsoft.com/es-es/library/dd409390.aspx.

27. **Netbeans. Netbeans.** [En línea] [Citado el: 4 de Diciembre de 2012.] http://www.netbeans.org/index_es.html.

28. **onsoni.** [En línea] [Citado el: 24 de Marzo de 2013.] http://www.onsoni.com/2011/11/definicion-del-codigo-fuente.html. 23.

29**. ormlite.** [En línea] [Citado el: 10 de Diciembre de 2012.] http://ormlite.com.

30. **Paz, Rosa María Torres de.** *El proceso de Ingeniería de Requisitos en el ciclo global del software.* Sevilla : s.n.

31. **Pressman, Roger S.** *Ingeniería del Software Un enfoque práctico 6ta Edición.* La Habana : Félix Varela, 2005.

32. **Pressman, Roger S.** *Ingeniería del Software, un enfoque práctico.* 2002.

33. **pruebasdesoftware.** [En línea] [Citado el: 25 de March de 2013.] http://pruebasdesoftware.com/laspruebasdesoftware.htm.

34. **Python.** [En línea] [Citado el: 10 de Diciembre de 2012.] http://www. python.org.ar/pyar/ORMs.

35. **R , JOHNSON y J, HOELLER.** *Expert One-on-One™ J2EE™ Development without EJB™.* (2004.).

36. **revistatelematica.** [En línea] [Citado el: 12 de Mayo de 2013.] revistatelematica.cujae.edu.cu%2Findex.php%2Ftele%2Farticle%2Fdownload%2F15%2F10&ei=DYOMUcnDOc-n4AOfhIGQAg&usg=AFQjCNEpYNL39MChwg7Ve1Ji1He-HGW1tg&cad=rja.

37. **Rosique, Francisca, Jiménez, Manuel y Sánchez , Pedro .** *Evaluación de herramientas de gestión de requisitos.* Cartagena : s.n.

38. **Sánchez Pescador, Elena.** *Análisis e implantación de una herramienta de gestión de requisitos para la gestión de servicios basado en la filosofía de ITL V3, CMMI de servicios y MOF.* 2009.

39. **Scribd.** [En línea] [Citado el: 25 de Noviembre de 2012.] http://es.scribd.com/doc/94927017/Ides.

40. **serk.kualtus.** [En línea] [Citado el: 25 de Marzo de 2013.] http://serk.kualtus.com/codigo.htm.

41. **Sites Google.** [En línea] [Citado el: 23 de Noviembre de 2012.] https://sites.google.com/site/programacion12se/conceptos-basicos-1.

42. **Slideshare.** [En línea] [Citado el: 7 de Diciembre de 2012.] http://www.slideshare.net/guestf131a9/herramientas-case.

43. **Sommerville, Ian.** *Software Engineering.* s.l. : Addison-Wesley, 2007.

44. **Sparx systems.** [En línea] [Citado el: 21 de Noviembre de 2012.] http://www.sparxsystems.com.ar/products/ea.html.

45. **Sparxsystems.** [En línea] [Citado el: 18 de Febrero de 2013.] http://www.sparxsystems.com.ar/uml-tutorial.html.

46. **Sparxsystems.** [En línea] [Citado el: 19 de Febrero de 2013.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.html.

47. **Sparxsystems.** [En línea] [Citado el: 21 de Noviembre de 2012.] http://www.sparxsystems.com.ar/products/ea.html.

48. **Sparxsystems.** [En línea] [Citado el: 25 de Marzo de 2013.] http://www.sparxsystems.com.ar/resources/tutorial/physical_models.html.

49. **umad.** [En línea] [Citado el: 21 de Febrero de 2013.] http://umad.zapatolibre.com/moodle_files/Articulo - El modelo relacional de bases de datos - Codd.pdf.

50. **Universidad de las Ciencias Informáticas.** *Guía de Trazabilidad.* La Habana : s.n. ., 2009.

51. **Universidad de las Ciencias Informáticas.** *Libro de Proceso para la Administración de Requisitos.* La Habana : s.n., 2009.

52. **upm.** [En línea] [Citado el: 18 de Febrero de 2013.] http://is.ls.fi.upm.es/docencia/proyecto/docs/patrones_gof.pdf.

53. **Visual Paradigm. Visual Paradigm .** [En línea] [Citado el: 10 de Diciembre de 2012.] http://www.visual-paradigm.com/product/vpuml/.

54. **Young, Ralph R.** *The Requirements Engineering Hand Book.* s.l. : Artech House, 2004.

55. **Zukowski, John.** *Java 2. J2SE 1.4.* La Habana : Félix Varela, 2007.