Universidad de las Ciencias Informáticas

Facultad 6

Sistema informático para la administración de requisitos en las pequeñas y medianas empresas de la Industria Cubana del Software

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

Autores: Yelena Ayala Pérez

Yanko Noriega Amado

Tutores: Ing. Enrique Pérez Rodríguez

Ing. Daimelys Piloto Garaboto

Ing. Dairys Febles Pérez

Co-tutora: MsC. Karina Pérez Teruel

La Habana, junio de 2011

Año 53 de la Revolución

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos 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 _____.

Yelena Ayala Pérez

_____
Firma del Autor

Yanko Noriega Amado

_____
Firma del Autor

Ing. Enrique Pérez Rodríguez

_____
Firma del Tutor

Ing. Dairys Febles Pérez

_____
Firma del Tutor

Datos de Contactos

DIPLOMANTES

Nombre: Yelena Ayala Pérez

E-mail: yayalap@estudiantes.uci.cu

Nombre: Yanko Noriega Amado

E-mail: ynoriega@estudiantes.uci.cu

TUTORES

Nombre: Dairys Febles Pérez

E-mail: dfebles@uci.cu

Nombre: Enrique Pérez Rodríguez

E-mail: enriquepr@uci.cu

Agradecimientos

Agradezco a Dios, a la Revolución y al líder de la Patria Fidel Castro Ruz.

A mis segundos padres, que son mis abuelitos (Juan y Mami Nela) les debo mi vida.

A mis papás, Adis y José Ángel, sepan que son los mejores padres de este mundo, a veces cuando sentía que el mundo venía abajo ustedes eran mi fuerza para seguir adelante por malo que fuese el camino por recorrer.

A mis hermanos Yunel y Yanelis por ser más que hermanos amigos, por ayudarme y quererme tanto. Los adoro a los dos.

A mi familia, y muy especialmente a mis tíos (tío Pedro, tía Juana, tío Salvy, tío Nene) por apoyarme en todo momento y confiar tanto en mi, mi meta ha sido siempre ser tan buena como lo son ustedes. Los quiero.

A mis tíos Ivo e Irela, que me ayudaron tanto que no tengo palabras para agradecerles.

A Carli (mi Chiti), que más que mi amigo ha sido mi hermano, mi protector, la persona en quien me he apoyado siempre, no puedo escribir lo tanto que te agradezco, tu eres para mi especial, te quiero mucho.

A todos mis primos les agradezco por contar siempre con el cariño que me han brindado, en especial a Yaremis, Yarelis, Alejandro, Nely, Manolito, Titi, Ernestico.

A mis vecinos por preocuparse y ocuparse tanto de mí, en especial a Maira, Olga, Ayannis, Daniel.

A mis amigos de estos 5 años de entrega y dedicación, a Yirina que ha sabido ser más que mi amiga, a Katia por enseñarme que ante situaciones difíciles uno crece, a Azalia por contar con su amistad, a la China que más que amiga ha sido hermana, a Yoennis por ser una cuñada especial.

A Leander la persona que me ayudó desde primer año, me dedicó su tiempo, y del cual aprendí mucho, te debo parte de lo que hoy soy.

A mi compañero de tesis y amigo desde primer año Yanko, te agradezco entrañablemente por aguantarme todo este curso, incluso cuando me estresaba, supiste darme todo el ánimo del mundo para seguir adelante, muchas gracias de todo corazón, siempre te llevaré presente.

A todas las personas que de una forma u otra me han ayudado, así como a todos los profesores que han contribuido en mi preparación.

Yelena Ayala Pérez

Agradecimientos

Agradezco con el amor más profundo e inmenso que se le puede brindar a una persona: a mí querida madre. Ella ha sido mi todo, sin ella no hubiera llegado donde estoy en estos momentos. A ti te lo debo todo…gracias por existir.

A mi abuela, que a pesar de nacer y nunca haberla conocido, ha sido de gran ayuda, apoyo, cariño y amor.

A mi madrina de bautizo (Lidia), realmente ha sido la segunda madre que muchos quisieran tener, de ti aprendí que los sueños nunca desaparecen siempre que las personas no los abandonan, este es mi gran sueño…

A Yaineris: parte de mi la has creado tú…y en general a toda su familia, en estos cinco años he descubierto que existen personas especiales y perfectas en este mundo…son ustedes, la verdad, de corazón muchas gracias, han sido y serán mi segunda familia…

A Alexander, Yasel, Lázaro, Erik, Randy han sido más que amigos los hermanos que nunca tuve.

A mi familia, mis amistades del barrio, de la escuela durante estos cinco años: Alain, Cecilia, Naridian, Ansel, Eugenio, Reynier, Evelín, Yadier, Dailenis, Gleydis, Marisniulkis de corazón gracias a todos.

A mis tutores, al tribunal, a Leander que ha sido imprescindible y a toda persona que ha estado presente en la formación de mi carrera y sobre todo en este tenso y último año.

Y por último y no por ser la menos importante a Yelena, a esta compañera de tesis y amiga de primer año, que ha sido la hermana que nunca tuve…es bonito decirlo realmente, gracias por tu apoyo, tu cariño, tu PACIENCIA; nunca cambies, espero que siempre seas tú todos los días de este mundo, esa personita tan especial que conocí en primer año…

Gracias Dios por hacer este sueño realidad.

Yanko Noriega Amado

Dedicatoria

Dedico este trabajo especialmente a dos personas a las cuales adoro con la vida, a los que supieron educar y formar a la numerosa familia que somos, a los que estuvieron conmigo desde que nací, mi adorado abuelo Juan (papá) y a mi querida abuela Nela (Mami Nela), que aunque ya no estés junto a nosotros siempre estas en mi corazón y en cada paso que doy. Hoy cumplo con lo que ustedes tanto deseaban.

A mis padres por todo el cariño, amor, dedicación, esfuerzo y consagración a lo largo de toda mi vida, por saber guiarme, educarme y formar en mí valores por los cuales hoy soy quien soy. Los quiero con todas las fuerzas de mi corazón.

A mí adorado hermano Yunel, tuyo también es este logro por quererme tanto al igual que yo a ti.

A mi familia en general.

Yelena Ayala Pérez

Consagro este trabajo a mi mamita linda por ser ella sola vine al mundo, por ser madre, padre, familia, todo, por educarme, enseñarme y guiarme en el camino que con los años marcarían quien realmente soy. Te quiero mucho hoy, mañana y siempre…gracias.

A mi abuela, que aún estando tan cerca y lejos a la vez siempre confió en mí…gracias de todo corazón.

Yanko Noriega Amado

Resumen

Atendiendo a la necesidad de las empresas de desarrollo de software en Cuba de obtener productos de calidad a la vez de mejorar la organización de sus procesos, y siguiendo la tendencia de las empresas de desarrollo de software en el mundo se recomienda la aplicación de un modelo de calidad para el mejor desempeño de sus procesos de desarrollo. Los modelos de calidad existentes presentan dificultades para establecerse en Cuba, porque son modelos privativos por lo cual requieren pago por concepto de licencias, capacitación y consultorías. Los estándares que coexisten como CMMI incluyen un área de procesos de administración de requisitos y como parte de la ingeniería de requisitos está la trazabilidad de requisitos. Por tal motivo el Centro de Calidad para Soluciones Informáticas (Calisoft) perteneciente a la Universidad de las Ciencias Informáticas (UCI) tiene la tarea de crear un modelo de calidad propio e implementar una suite de herramientas que permita automatizar las prácticas. Entre estas herramientas se propone la creación de una herramienta para la trazabilidad de requisitos. En el presente trabajo se desarrolló un sistema informático para lograr la automatización o flexibilización del proceso de trazabilidad de requisitos. Para ello se siguió el Proceso de Desarrollo Unificado (RUP) como metodología de software, se usó UML como lenguaje de modelado y la herramienta CASE Visual Paradigm para UML y como lenguaje de programación Java, todas en general constituyen herramientas de software libre.

Palabras Claves

Modelo de Calidad, Trazabilidad Bidireccional de Requisitos, Ingeniería de Requisitos.

Índice de Contenido

Índice de tablas

INTRODUCCIÓN

En el actual contexto competitivo de las empresas de software, la calidad es una necesidad prioritaria en el mercado. Existen normas específicas para medir la calidad de los productos que ayudan a las empresas a certificar y comparar sus resultados contra los requisitos establecidos. (1) Modelos de calidad como CMMI, ISO o Six Sigma tratan de instituir procesos de desarrollo de software estables que sean independientes del conocimiento individual de los ingenieros de software.

Con el objetivo de alcanzar la madurez que requiere la producción de software, se mantienen por parte de los profesionales de esta rama, esfuerzos continuados por modelar y estandarizar las mejores prácticas de calidad; así como mejorar y facilitar el trabajo en cada área de procesos del modelo. Para ello es necesario definir modelos y estándares que mejoren continuamente los procesos, luego de implantados en las organizaciones, esto es algo complejo que requiere tiempo, esfuerzo y la inversión de grandes sumas de dinero.

Cuba, a la par de este progreso y dado el creciente y cada vez más acelerado avance de la informática, ha impuesto el perfeccionamiento de un Modelo de Calidad que brinde a la Industria Cubana de Software una guía práctica para el mejor desempeño de sus procesos de desarrollo.

El Centro de Calidad para Soluciones Informáticas (Calisoft) perteneciente a la Universidad de las Ciencias Informáticas (UCI) y al Ministerio de la Informática y las Comunicaciones (MIC), es el principal responsable de la creación de este Modelo de Calidad. En este sentido han surgido diversos temas de investigación encaminados a tomar las mejores prácticas de los modelos existentes y adaptarlas a las necesidades propias de la Industria Cubana de Software.

Dentro de las áreas claves identificadas y que formará parte del modelo cubano, se encuentra la Ingeniería de Requisitos, y como técnica la trazabilidad bidireccional de requisitos.

Los modelos estudiados y por tanto sus procesos relacionados con la Ingeniería de Requisitos (IR) presentan los siguientes inconvenientes para su aplicación en la Industria Cubana del Software.

Son diseñados para grandes empresas de desarrollo de software, presentan mucha complejidad para su entendimiento, y tienen altos costos de consultoría y certificación.

Por otra parte las herramientas de gestión de requisitos existentes en el mundo, no se adaptan a las necesidades del nuevo modelo cubano, debido a que presentan desventajas como:

La mayoría de estas herramientas son software propietario, lo cual no se corresponde con el principio de independencia tecnológica en el cual está basado actualmente el desarrollo de software en Cuba, por lo que no es factible su uso por la licencia que requieren y en su totalidad son herramientas de escritorio que no pueden interconectarse a diferentes estaciones de trabajo.

A partir de los elementos expuestos se plantea como **problema científico**: ¿Cómo contribuir en la ejecución de una herramienta para la administración de requisitos en las pequeñas y medianas empresas (PyMEs) de la Industria Cubana del Software?

Se define como **objeto de estudio**: El proceso de administración de requisitos enmarcado en el **campo de acción:** Técnica de trazabilidad bidireccional para el proceso base de ingeniería de requisitos en el Modelo Cubano de Calidad.

Para resolver el problema se trazó como **objetivo general**: Desarrollar un sistema informático para la administración de requisitos en las pequeñas y medianas empresas de la Industria Cubana del Software.

A partir del análisis del objetivo general se derivan los siguientes **objetivos específicos**:

- Identificar las funcionalidades que debe cumplir el sistema informático.
- Diseñar las funcionalidades especificadas para el sistema informático.
- Implementar las funcionalidades del sistema informático.
- Realizar pruebas al sistema informático.

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

1- Análisis de los principales conceptos y herramientas relacionadas con el proceso de administración de requisitos y la técnica de trazabilidad bidireccional de requisitos.

2- Estudio y selección de:

- La metodología de desarrollo.
- Los lenguajes de modelado y de programación a utilizar

- Las herramientas para desarrollar el sistema.
- El framework de desarrollo.

3- Identificación de las fuentes de los requisitos para la herramienta de trazabilidad.

4- Obtención de los requisitos.

5- Definición de los requisitos.

6- Especificación de los requisitos (obtener el documento Especificación de requisitos de software).

7- Validación de los requisitos (línea base de requisitos).

8- Definición de los casos de uso del sistema.

9- Descripción de los casos de uso del sistema.

10- Confección del diagrama de clases del diseño.

11- Confección de los diagramas de interacción del diseño.

12- Diseño de la base de datos del sistema informático.

13- Realización de los diagramas de componentes del sistema informático.

14- Implementación de los componentes del sistema informático.

15- Realización de las pruebas a nivel de desarrollador del sistema informático.

En cuya solución se utilizarán como **métodos científicos**:

Métodos empíricos:

Entrevistas: A miembros de Calisoft con experiencia en el nuevo modelo cubano para la comprensión del problema existente, ya que la solución propuesta es un componente de dicho modelo.

Métodos teóricos:

Analítico-Sintético: Para resumir, enunciar y describir los requerimientos del nuevo modelo cubano para el desarrollo de software.

Análisis Histórico-Lógico: Se utiliza para hacer un estudio del estado del arte de las herramientas existentes para gestionar la trazabilidad de requisitos y sus usos en el ámbito nacional e internacional, así como las ventajas y desventajas que poseen.

Como posible **resultado** de la investigación se obtendría: La implementación de una herramienta para la administración de requisitos.

El contenido de este documento está estructurado de la siguiente manera:

Capítulo 1: Fundamento teórico: Se argumentan las características principales de la metodología de desarrollo utilizada y otras herramientas en las que se apoya el desarrollo de sistema.

Capítulo 2: Características del sistema. En este capítulo se realiza el diagrama del modelo conceptual propuesto y se definen los requerimientos a implementar. Además, se realiza el diagrama de casos de usos del sistema y la descripción de los mismos.

Capítulo 3: Diseño del sistema: Se explican los patrones de desarrollo de software usados, se presentan los diagramas de clases del diseño y los diagramas de interacción. Finalmente se muestra el diagrama de despliegue, con el objetivo de tener una idea más clara de cómo quedará el sistema una vez desplegado.

Capítulo 4: Implementación y Pruebas: Se muestran los diagramas de componentes que se definieron durante la implementación del sistema, se describen implementaciones relevantes y se visualizan algunas pantallas del sistema informático. Además, se detallan pruebas realizadas sobre la aplicación para comprobar sus funcionalidades.

1. CAPÍTULO 1: FUNDAMENTO TEÓRICO

1.1 Introducción

En este capítulo se realiza un estudio del estado actual de la Ingeniería de Requisitos (IR), la trazabilidad bidireccional de los requisitos y las herramientas asociadas; así como las comparaciones de las Metodologías, Lenguajes de Programación, Entornos de Desarrollo Integrado (IDE), Gestores de Base de Datos y Framework, para luego realizar la selección de lo que se utilizará en el desarrollo de la aplicación.

1.2 Ingeniería de Requisitos

La IR cumple un papel primordial en el proceso de producción de software, ya que se enfoca en un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, las necesidades de los usuarios o clientes; de esta manera, se pretende minimizar los problemas relacionados con la mala gestión de los requerimientos en el desarrollo de sistemas.

Dentro del estudio que se llevó a cabo para la realización del presente trabajo de diploma, se encontraron varios conceptos relacionados con el tema de ingeniería de requisitos, propuesto por diferentes autores, según su punto de vista.

Según Amador Duran Toro en su tesis de doctoral plantea que la Ingeniería de Requerimientos es:” El proceso de estudiar las necesidades del usuario para llegar a una definición de requisitos de sistema, hardware o software”. (2)

Desde otra perspectiva la compañía IEEE Standard Glossary of Software Engineering Terminology define que un requerimiento es:” Una condición o capacidad que debe estar presente en un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación u otro documento formal.” (3)

O simplemente “Es una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.” (3)

Cada autor tiene una forma de conceptualizar los requerimientos o requisitos, la investigación estará enmarcada en el siguiente concepto:

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

1.2.1 Técnica trazabilidad bidireccional de requisitos en el proceso de ingeniería de requisitos

La trazabilidad en la Ingeniería de Software es una práctica de control que ayuda a obtener el producto en el dominio de la solución lo más exacto y fiable posible a las necesidades expresadas por el cliente en el dominio del problema. Es la base de la gestión de los requisitos, puesto que brinda la información necesaria para su control y soporte a lo largo del proceso de desarrollo de software. En otras palabras, posibilita la verificación de la transformación de los requisitos en elementos de modelos sucesores, así como el análisis y gestión del cambio en ellos, y verifica su completitud y coherencia.

Dentro de los conceptos más vistos en la actualidad referente al tema de la técnica trazabilidad bidireccional de requisitos, según Patricio Letelier expresa que: “La trazabilidad bidireccional de requisitos no es más que la habilidad para seguir la vida de un requisito en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de las especificaciones generadas durante el proceso de desarrollo.” (4)

Por otra parte Víctor Anaya en su informe en conjunto con el autor del anterior concepto plantea en su escrito que la trazabilidad de requisitos:

Se define como el arte de describir y seguir la vida y el estado en todo momento de un requisito en los dos sentidos, es decir, lo mismo hacia el momento de su captura o definición que hacia la etapa de implementación. (5)

Los autores de esta investigación como parte del trabajo consideran que la trazabilidad de requisitos no es más que la capacidad de describir y de seguir la vida de un requisito, desde sus orígenes, a través de su

desarrollo, especificación, despliegue y uso subsecuente, así como a lo largo de períodos de refinamiento y de la iteración en curso en cualquiera de las fases que se encuentre.

1.2.2 Herramientas existentes en el mundo para la trazabilidad de requisitos

Para analizar algunas herramientas de gestión de requisitos, la presente investigación se fundamenta en la encuesta realizada por INCOSE. Se seleccionaron 9 de las herramientas que cumplen con la mayoría de las funciones que podrían esgrimirse para la realización del sistema a desarrollar en el presente trabajo. Dentro de estas se encuentran:

- RequisitePro
- Borland CaliberRM
- Doors Telelogic
- Gatherspace
- REM
- OSMRT

Algunas características de estas herramientas son las siguientes**:**

RequisitePro:

Esta herramienta, desarrollada por una de las compañías más importantes dentro del campo de la informática, se considera una de las herramientas más completas y potentes dentro del análisis y la gestión de los requisitos. Una de las grandes ventajas que aporta este producto es la compatibilidad existente entre su software y algunos de los programas más utilizados.
Por ejemplo, es capaz de comunicarse eficientemente con el Microsoft Word, de manera que la realización de los informes es más sencilla al tiempo que se ofrece al usuario una interfaz conocida para el desarrollo de su labor. Además de esta compatibilidad, el programa también se comunica con gran eficiencia con algunos de los sistemas de bases de datos más utilizados en el mundo de la informática (DB2 de IBM, Microsoft SQL Server, Microsoft Access y Oracle) de manera tal que se controla el acceso a los datos

existentes en el sistema al mismo tiempo que se tiene un repositorio central de datos. Como desventaja de esta herramienta es que es sólo para Windows, propietario y su licencia en el mercado está muy elevada. Además tiene que estar instalado en cada computadora que se vaya a utilizar porque de otra forma no se obtiene sus beneficios.

Borland CaliberRM:

Herramienta creada por Borland para facilitar la definición, el análisis y la gestión de los requisitos del software, mejora en su calidad y reduce el nivel de fracaso. Es famosa fundamentalmente por su facilidad de uso, puesto que permite una rápida adopción por el personal no técnico y además, actualmente está muy bien posicionada en el mercado. Dentro de ella, los requisitos se pueden clasificar en varios paquetes como requisitos de negocio, de usuario, funcionales, de diseño y escenarios de prueba, entre otros.

Dispone de un repositorio (vista “grid” de requisitos) centralizado y seguro de manera que todos los requisitos de un proyecto estén localizados. Así se gestionan a través del ciclo de vida de la aplicación, mejora la comunicación entre los diferentes participantes y facilita la detección temprana de posibles errores. Proporciona múltiples métodos para la visualización de la trazabilidad, que ayudan a los usuarios a comprender inmediatamente el alcance del análisis requerido para calibrar el impacto del cambio de los requisitos.

Por último, se exponen varios inconvenientes sobre la herramienta. Lo primero a destacar es su alto precio de licencia. Por otra parte, adquiere una debilidad significativa al tener que ser instalada en cada máquina en la que vaya a ser utilizada. Esto hace más complicada la gestión de software, lo que dificulta las actualizaciones del mismo. (6)

DoorsTelelogic:

Esta herramienta, destacada en el mercado, forma parte de una familia de soluciones que mejoran la calidad y optimizan la comunicación y la colaboración. Se encarga de gestionar los requisitos, enlazar, trazar, analizar y manejar un amplio rango de información para asegurar la adecuación del proyecto a los requisitos y estándares.

DoorsTelelogic proporciona las siguientes funcionalidades:

- Importar/exportar: Permite importar y exportar información mediante Microsoft Word, Excel, PowerPoint, Outlook, textos planos (ASCII), hojas de cálculo y RTF[1], entre otros.
- Trazabilidad: Dispone de una matriz de trazabilidad de requisitos fácil de usar y actualizar. Se encarga de mantener y gestionar las dependencias entre los distintos requisitos mediante el establecimiento de una trazabilidad entre ellos.
- Flexibilidad en el tamaño del proyecto, lo que permite aceptar un número ilimitado de usuarios.

Por último se ha de mencionar que aunque la versión instalable de la herramienta es completa e impecable, la versión Web de ella deja mucho que desear. Solamente tiene disponible una mínima parte de las funcionalidades totales que presenta la herramienta en sí, lo que provocará una incompleta satisfacción de las necesidades que una adecuada gestión de requisitos es capaz de originar. (6)

Gatherspace:

Herramienta Web capaz de editar, compartir y distribuir requisitos on-line. 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 puesto que 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.

Como inconvenientes importantes de la herramienta se enfatiza que el manejo de ella, al ser una herramienta web, puede ser lento puesto que está a expensas del servidor. Este se puede encontrar caído o sobrecargado lo que ralentizará o restringirá el trabajo en el proyecto. (6)

REM:

El nombre completo de la herramienta es Requisite Management desarrollada por la Universidad de Sevilla. Es una herramienta experimental gratuita de Gestión de Requisitos diseñada para soportar la fase

[1] Formato de Texto Enriquecido (RTF). Es un formato de archivo informático desarrollado por Microsoft en 1987 para el intercambio de documentos multiplataforma. La mayoría de los procesadores de textos son capaces de leer y escribir documentos RTF.

de Ingeniería de Requisitos de un proyecto de desarrollo software, de acuerdo con la metodología definida en la Tesis Doctoral "Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información", realizada por el profesor Amador Durán del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Dentro las principales funcionalidades se pueden mencionar:

- Gestión de requisitos, diferenciando entre:
    - Trazabilidad entre todos los documentos de trabajo.
- Personalización y configuración.
- Descripción de un requisito mediante la secuencia de pasos de su caso de uso.
- Gestión de la configuración.
- Herramientas de migración para los diversos cambios de versiones.

Los inconvenientes que se observaron son los siguientes:

- No es una herramienta Open Source y en principio el autor no plantea liberar su código.
- No incorpora gestión de versiones.
- No es posible controlar la numeración de los requisitos.
- No se puede añadir formato al texto. (7)

OSMRT:

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).

Los diversos módulos integrados en la herramienta son:

- Administración y Configuración: Se van a definir los atributos de las diferentes funcionalidades, requisitos y casos de prueba. Además cada usuario fijará valores predefinidos (Valores por defecto de prioridades, estados, versiones,..) y personalizará sus vistas.
- Gestión de documentos de ingeniería de requisitos.

- Trazabilidad entre documentos de trabajo: La trazabilidad se produce entre requisitos, frente a requisitos, funcionalidades o casos de prueba. 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.
- Informes y estadísticas: Los informes y estadísticas creados podrán ser básicos o específicos, con la posibilidad de ser exportados a HTML o PDF.

Las principales divergencias que se examinaron son los siguientes:

- Es un desarrollo llevado a cabo por una persona individual, por lo que existe el riesgo de que no sea sostenible a lo largo del tiempo.
- No existe un soporte empresarial.
- Las nuevas versiones no están planificadas ni se anuncian claramente las mejoras que serán incorporadas. Es posible que las nuevas versiones no sean compatibles con las anteriores.
- No es posible generar automáticamente un documento de requisitos para entregar al cliente·
- Algunas funcionalidades no han sido desarrolladas completamente.
- La interfaz de usuario es lenta en ocasiones. (6)

Después de analizar las herramientas y de conocer las funcionalidades que cada una ofrece, se pueden identificar los requerimientos funcionales y no funcionales que tendrá la herramienta que guía la presente investigación, adaptada a las necesidades del nuevo modelo cubano. Por todo lo antes expuesto para la realización de la herramienta de este trabajo, será un punto clave saber que dicho sistema será de código abierto, que en el mercado sólo existe una que es bien usada en el mundo, pero aún le falta mejorar aspectos que se tomarán en cuenta para este software.

1.3 Metodologías de Desarrollo de Software

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto software, se definen una serie de pasos a seguir para obtener un software de calidad. Las metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas abarcan procedimientos, técnicas, documentación

y herramientas que se utilizan en la creación de un producto de software. Sus procesos se descomponen a nivel de tareas o actividades elementales, donde cada tarea está identificada por un procedimiento que define la forma de llevarla a cabo. (8)

1.3.1 Rational Unified Process (RUP)

El Proceso Unificado Racional, Rational Unified Process en inglés, y su sigla RUP, es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. (8)

El ciclo de vida de RUP se caracteriza por**:**

- Estar dirigido por casos de uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).
- Ser centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.
- Ser iterativo e incremental: Propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son mini proyectos.

Fases:

- Concepción o inicio: Comprender los requisitos y determinar visión y alcance del proyecto.
- Elaboración: Asignar recursos, especificar las características y definir la arquitectura.
- Construcción: Implementación, construir el producto operacional.
- Transición: Hacerlo operativo para los usuarios, nivel correcto de calidad para entregar.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo.

Flujos de trabajo:

Modelado del negocio**:** Describe los procesos de negocio, e identifica quiénes participan y las actividades que requieren automatización.

Requerimientos**:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

Análisis y diseño**:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.

Implementación**:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.

Prueba (Testeo): Busca los defectos a lo largo del ciclo de vida.

Administración del proyecto**:** Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

Administración de configuración y cambios**:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

Ambiente**:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

RUP se robustece con la documentación, que aunque se ve afectada con los posibles cambios volátiles que los clientes soliciten en cuanto a funcionalidades del software, representa una gran ventaja, ya que gracias a su plan de desarrollo se pueden reconocer los problemas y fallos de forma temprana y corregirlos. Pero además, no es un sistema con pasos firmemente establecidos, sino una metodología adaptable al contexto y necesidades.

1.3.2 Extreme Programming (XP)

XP (eXtreme Programming) nace como nueva disciplina de desarrollo de software hace aproximadamente unos seis años, y ha causado un gran revuelo entre el colectivo de programadores del mundo. Kent Beck, su autor, es un programador que ha trabajado en múltiples empresas y que actualmente lo hace como programador en la conocida empresa automovilística DaimlerChrysler. Con sus teorías ha conseguido el respaldo de gran parte de la industria del software y el rechazo de otra parte. La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código; para algunos no es más que aplicar una pura lógica. (9)

Los objetivos de XP son muy simples: la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, debe responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final de ciclo de la programación. El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, como los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo de software. (9)

Consta de cuatro actividades básicas: codificar, hacer pruebas, escuchar y diseñar; y de cuatro fases que son: la planificación, el diseño, el desarrollo y las pruebas.

1.3.3 OpenUP

Carmina Lizeth Torres Flores y Germán Harvey Alférez Salinas en un escrito acentúan que OpenUP es un proceso de desarrollo iterativo del software que es mínimo, completo, y extensible. El proceso es mínimo porque solamente el contenido fundamental es incluido; es completo puesto que puede ser manifestado como todo el proceso para construir un sistema; extensible a causa de que puede ser utilizado como

fundamento sobre el cual el contenido de proceso se pueda agregar o adaptar según lo necesitado. Como metodología de desarrollo es conducida por el principio de colaboración para alinear intereses y para compartir su comprensión. (10)

Es una versión más ágil de lo que es RUP. OpenUP nos plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo corto. Principalmente plantea que se deben utilizar solo los procesos que sean necesarios y en este caso se necesita de personas y profesionales que sean capaces de distinguir entre lo necesario y lo que no es necesario para que el proyecto no tenga errores y con eso evitar entregar al usuario final un producto de mala calidad. Además esboza que no se deben utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario y puede ser modificado, mejorado y extendido.

Metodología de desarrollo seleccionada

Después de analizar estas tres metodologías de desarrollo de software se decidió utilizar RUP, ya que es la que más se adapta a las necesidades del equipo de desarrollo y en la que más experiencia se tiene. Además constituye un marco de trabajo genérico que puede ser adaptado a una gran diversidad de sistemas de software independientemente del tamaño del proyecto, el tipo de organización y los diferentes niveles de aptitud. Es una metodología adaptable al contexto y a las necesidades que permite tener bien claro y accesible el proceso de desarrollo que se quiere alcanzar.

1.4 Lenguaje de Modelado

Debido a la selección de la metodología de desarrollo de software se hace necesario utilizar como lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language). UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Según Enrique Orallo precisa que “es un lenguaje gráfico para, especificar, construir y documentar un sistema”. (11)Y Craig Larman en su libro de “UML y Patrones” especifica que el mismo brinda un estándar para describir un plano del software, incluye aspectos conceptuales tales como procesos de negocios, funciones del sistema, expresiones de lenguajes de programación, esquemas de bases de datos y componentes de

software reutilizables. (12) En concreto ayuda a comprender y a mantener de una mejor forma un sistema basado en un área que el analista o desarrollador puede desconocer.

La decisión de utilizar UML como notación para el desarrollo del software se debe a que se ha convertido en un estándar que tiene las siguientes características:

- Permite modelar sistemas haciendo uso de técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- Es independiente del proceso, aunque para utilizarlo óptimamente se debería emplear en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

1.5 Herramienta para el modelado

Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costes, implementación automática de parte del código con el diseño dado, compilación automática, documentación o detección de errores entre otras. (8)

1.5.1 Visual Paradigm

Visual Paradigm es una herramienta CASE que usa UML como lenguaje de modelado. La misma soporta, completamente, el ciclo de vida del desarrollo de software: Análisis y Diseño Orientados a Objetos, Construcción, Pruebas y Despliegue. También permite la generación automática de reportes en formato pdf y html, el reconocimiento de artefactos de ingeniería a partir de reconocimiento de textos, implementa una actualización automática del modelo de diseño y código permitiendo mantener la documentación de ambos modelos actualizadas con los cambios que ocurran en ambos sentidos, optimizando la descripción textual de elementos de código a partir de la descripción visual. Es capaz de importar y exportar elementos de otras herramientas CASE como Rational Rose y presenta una interfaz con increíbles facilidades a la hora de modelar los diagramas que soportan la Ingeniería de Requisitos (13).

Dentro de las principales características de la herramienta están:

- Soporte de UML versión 2.0.
- Disponibilidad de integrarse en los principales IDEs.
- Soporta una gama de lenguajes en la Generación de Código e Ingeniería Inversa en Java, C++, CORBA IDL, PHP, Ada y Python.
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio.
- Diagramas de flujo de datos.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas. Importación y exportación de ficheros. Editor de figuras.

1.5.2 Rational Rose

En el sitio web “Buenas Tareas” se precisa que Rational Rose es actualmente conocida como una familia de software de IBM para el despliegue, diseño, construcción, pruebas y administración de proyectos en el proceso desarrollo de software. Brinda soporte al modelado visual, permitiendo hacer énfasis en los detalles más importantes, centrándose en los casos de uso y enfocándose hacia un software de mayor calidad, empleando un lenguaje estándar común, que facilita la comunicación. Es la herramienta más poderosa del mercado, basado en eclipse. (14)

Con el uso del lenguaje común de modelado UML, se facilita el trabajo para el equipo de desarrollo y se garantiza mayor eficiencia y calidad del trabajo. Permite generar códigos en diferentes lenguajes de programación partiendo de modelado UML. Además, hace posible efectuar la ingeniería inversa, es decir, obtener información del diseño de un software partiendo de su código. Pero presenta algunas decadencias ya que necesita de mucha memoria para poder de alguna forma ser manejado de forma rápida y eficiente. La mayoría de los lenguajes orientados a objetos imponen una carga bastante pesada a la computadora, además de que son caros.

1.5.3 ArgoUML

Erik Alexander Benítez en un trabajo investigativo define que ArgoUML es una herramienta para modelar sistemas, mediante la cual se realizan diseños en UML "Lenguaje de Modelado Unificado", basada en Java. Puede crear la mayoría de los diagramas estándares de UML. Nace como proyecto de Jason Robbins. El nombre "Argo" se refiere a la biblioteca de las características cognoscitivas de Argo que fue concebida como una herramienta para el análisis y el diseño de los sistemas de software orientados al objeto. En este sentido es similar a muchas de las herramientas comerciales CASE que se venden como herramientas para modelar sistemas de software. (15)

Características de ArgoUML:

- Las ayudas de esta herramienta de modelado abren extensiones UML, XMI, SVG, OCL y otros de los estándares. En este aspecto, todavía está delante de muchas herramientas comerciales.
- Es 100% puro de Java, esto permite que funcione en todas las plataformas para las cuales un puerto confiable de la plataforma Java2 esté disponible.
- Es un proyecto abierto que permite que la disponibilidad del código fuente asegure de que una nueva generación de diseñadores y de investigadores de software tenga un marco probado del cual puedan conducir el desarrollo y la evolución de las tecnologías de la herramienta CASE.

Pero presenta también decadencias que no permiten utilizarla en este trabajo: tiene una instalación costosa; es poco amigable y es muy difícil de interactuar con ella. (15)

Herramienta seleccionada para el modelado

Visual Paradigm es la herramienta CASE que se utilizará en el modelado de esta herramienta debido a que es multiplataforma, además de ser una potente herramienta CASE con licencia libre para uso de la comunidad de desarrolladores. Permite modelar UML, con 13 tipos diferentes de diagramas que pueden ser exportados como imágenes en formato JPG, PNG, entre otros. Proporciona a los desarrolladores una plataforma con interfaz amigable que les permite diseñar un producto con calidad de forma muy rápida. Puede ser extendido, pues presenta soporte al diseño personalizado, lo que permite incorporar nuevas formas y notaciones, mediante el uso de imágenes o íconos importados.

1.6 Entorno de Desarrollo Integrado (IDE)

Un Entorno Integrado de Desarrollo en inglés Integrated Development Enviroment (IDE) es un programa compuesto por un conjunto de herramientas para un desarrollador que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Dentro de los más populares a nivel mundial se encuentran el Eclipse y el NetBeans. (16)

1.6.1 Eclipse

Es un IDE libre, multiplataforma y de licencia Open Source. El mismo fue creado por IBM, como sucesor de la familia VisualAge. La base arquitectónica para el mismo es la plataforma de cliente enriquecido, todo lo contrario de cliente ligero y está constituido por varios componentes entre los cuales se encuentran:

- Plataforma principal - Inicio de Eclipse y ejecución de plugins.
- OSGi - Una plataforma para el desarrollo estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace es un manejador de texto y de archivos, incluye editores de texto.
- El Workbench de Eclipse, posee vistas, editores, perspectivas y asistentes.

Características de Eclipse

- Estructura de plugins que hace sencillo añadir nuevas características y funcionalidades.
- Fácil vinculación con herramientas de control de versiones como Subversion, Git y Bazaar entre otros.
- Asistentes para la creación, exportación e importación de proyectos.

1.6.2 NetBeans

Herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrita en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender del IDE NetBeans. Es un producto libre y gratuito sin restricciones de uso. (17)

Soporta varios lenguajes, entre ellos:

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

Características de NetBeans:

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

IDE de desarrollo seleccionado

El entorno de desarrollo integrado que fue seleccionado por el colectivo de autores para desarrollar el software, fue el Netbeans, en su versión 6.9, por ser de código abierto, por su factibilidad de uso, además de considerarse más estable y rápido; su diseño global permite tener las herramientas que necesitan los programadores inmediatamente al alcance de sus manos. Es fácilmente integrable con la herramienta CASE Visual Paradigm.

1.7 Lenguaje de Programación

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se pone a disposición del programador para que éste pueda comunicarse con los dispositivos hardware y software existentes. (18)

1.7.1 PHP 5

PHP (Hypertext Preprocessor) es un lenguaje script para el desarrollo de páginas web dinámicas del lado del servidor, cuyos fragmentos de código se intercalan fácilmente en páginas HTML, debido a esto, y a que es de Open Source (código abierto), es el más popular y extendido en la web. (19)

PHP tiene muchas ventajas algunas de ellas son:

- Multiplataforma: Inicialmente fue diseñado para entornos UNIX por lo que ofrece más prestaciones en este sistema operativo, pero es perfectamente compatible con Windows.
- Soporte para varios servidores Web. Mucha documentación (Ejemplos, manuales).
- Posee una sintaxis bastante clara.
- Fácil aprendizaje.
- Es seguro.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Además está orientado a objetos.

1.7.2 JAVA

Es un lenguaje de programación sencillo, orientado a objetos, de propósito general e independiente de la plataforma de desarrollo. Fue creado por James Gosling, cumpliendo así con el slogan de la compañía Sun Microsystems "compilar una vez y ejecutar en cualquier parte". (20)

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

Las características principales de Java respecto a otros lenguajes de programación son: (21)

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

Lenguaje de programación seleccionado

Teniendo en cuenta las características antes mencionadas de Java, especialmente por ser multiplataforma sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Además de que en la actualidad es muy utilizado ya que permite el desarrollo de programas seguros y de alto rendimiento, el colectivo de autores decide utilizarlo como lenguaje de programación.

1.8 Frameworks

En el desarrollo de software, un framework es una estructura conceptual y tecnológica de soporte definida, normalmente, con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En el desarrollo de la arquitectura base del sistema se utilizó el framework de aplicación Spring, Spring-Security y el framework de soporte Hibernate, ambos son muy populares por sus múltiples ventajas en el desarrollo de aplicaciones web y son de código abierto y por último Ext JS, utilizado como framework JavaScript o de presentación para crear interfaces de manera fácil.

1.8.1 Spring 3.0.2

Spring es un framework de aplicación desarrollado por la compañía Interface 21, para aplicaciones escritas en el lenguaje de programación Java. Sus desarrolladores basados en sus experiencias en el desarrollo de aplicaciones J2EE (Java 2 Enterprise Editions), incluyendo EJB (Enterprise JavaBeans), Servlets y JSP (Java Server Pages), lograron combinar dichas herramientas y otras más en un solo paquete, para brindar una estructura más sólida y un mejor soporte para este tipo de aplicaciones. (22)
Es un framework basado en la Inversión de Control (IoC) y en la Programación Orientada a Aspectos (AOP). Se distribuye de forma libre y su código es abierto. Ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar.

La simplificación del desarrollo de aplicaciones y de sus respectivas pruebas es una de las claves de su éxito. (22)

Arquitectura de Spring

Spring es un framework modular que cuenta con una arquitectura dividida en aproximadamente veinte módulos integrados en capas (Fig.1.1), lo cual permite tomar y ocupar únicamente las partes que interesen para el proyecto y juntarlas con gran libertad. (22)

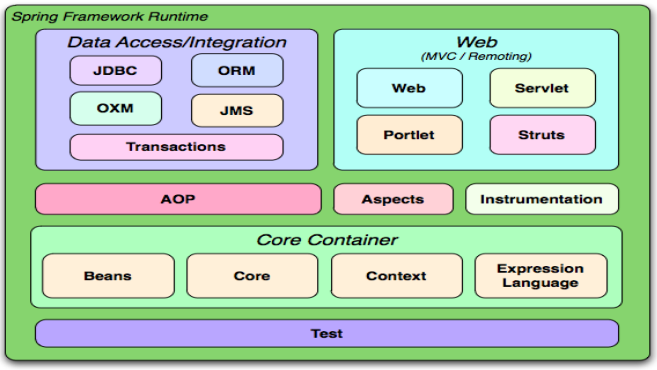

Figura 1: Arquitectura de Spring 3.0.2

Capa Core Container

Esta parte es la que provee la funcionalidad esencial del framework, está compuesta por el Core, Beans, Context, and Expression Language.

- El **Core y Beans** proveen las partes fundamentales del framework, e incluye la Inversión de Control (IoC) y las opciones de inyección de dependencias.

Capa Acceso a Datos/Integración

Contiene JDBC (Java Database Connectivity), ORM (Object-Relational Mapping), OXM (Mapeo Object/XML), JMS (Java Message Service) y los módulos de transacciones.

- El módulo **JDBC** proporciona una abstracción del modelo JDBC y elimina la necesidad de usar el JDBC básico que proporciona JAVA para acceder a la base de datos y controlar los errores.
- El módulo de **ORM** proporciona una capa de integración para las API's de mapeo entre objetos y el modelo relacional. Incluye integración con Hibernate, Java Persistence API (JPA), Java

Data Objects (JDO) e iBatis. Este módulo permite integrar los anteriores frameworks con la funcionalidad y las características que ofrecen Spring.

Capa Web

La capa Web contiene los módulos Web, Web-Servlet, Web-Struts, y Web-Portlet.

- El módulo **Web-Servlet** contiene el Modelo-Vista-Controlador de Spring (MVC) y la ejecución de aplicaciones web. Spring MVC, establece una clara separación entre las diferentes capas, y se integra con todas las otras características del Spring Framework.

Capa AOP (Aspect-Oriented Programming)

El objetivo del módulo no es proveer la implementación más completa posible, sino una integración cercana entre la implementación de AOP y el contenedor de aplicaciones para resolver los problemas comunes de las aplicaciones empresariales.

Capa Test

La capa de prueba apoya el examen de los componentes de Spring con JUnit o TestNG. Proporciona carga constante de ApplicationContexts y el almacenamiento en caché de los contextos.

A continuación, una representación gráfica del funcionamiento del MVC en Spring (Fig.1.2)

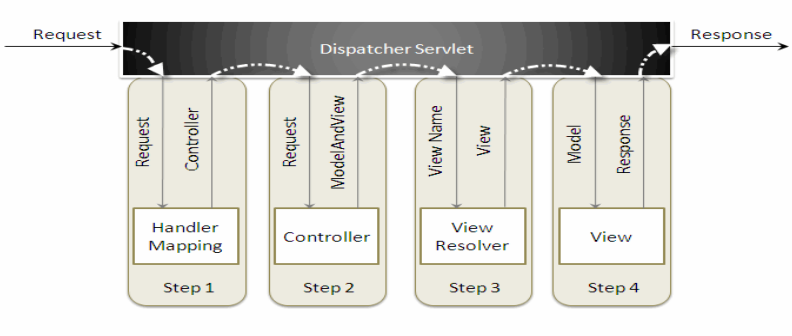

Figura 2: MVC en Spring.

Spring-Security 3.0.5

Spring Security es una autenticación potente, altamente personalizable y un marco de control de acceso. Es uno de los proyectos de primavera más maduros y ampliamente utilizados. Fundada en 2003 y mantenido activamente por SpringSource, ya que, hoy en día se utiliza para asegurar numerosos entornos exigentes incluyendo agencias gubernamentales, las aplicaciones militares y los bancos centrales. Es también fácil de aprender, implementar y administrar. (23)

Dentro de las ventajas principales que tiene este framework se pueden mencionar que:

- Es capaz de gestionar seguridad en varios niveles: URLs que se solicitan al servidor, acceso a métodos y clases Java, y acceso a instancias concretas de las clases.
- Permite separar la lógica de nuestras aplicaciones del control de la seguridad, utilizando filtros para las peticiones al servidor de aplicaciones o aspectos para la seguridad en clases y métodos.
- La configuración de la seguridad es portable de un servidor a otro, ya que se encuentra dentro del WAR o el EAR de nuestras aplicaciones.
- Soporta muchos modelos de identificación de los usuarios (HTTP BASIC, HTTP Digest, basada en formulario, LDAP, OpenID, JAAS y muchos más). Además podemos ampliar estos mecanismos implementando nuestras propias clases que extiendan el modelo de Spring Security. (23)

1.8.2 Framework JavaScript

Ext JS 3.3.1

Ext JS es una librería JavaScript para la creación de aplicaciones enriquecidas del lado del cliente. Sus características principales son: gran desempeño, componentes de interfaz de usuario personalizables, con buen diseño y documentación.

Ext JS tiene un modelo de licencia dual. Para aplicaciones web comerciales hay que adquirir una licencia y para aplicaciones web open source, que sean compatibles con GNU GPL license v3, se puede utilizar la licencia gratuita. Como la solución que se propone no es con fin comercial, se utiliza la licencia gratuita.

1.8.3 Hibernate-JPA (Java Persistence API)

Hibernate es un framework ORM para la plataforma Java disponible además para la plataforma .NET con el nombre de NHibernate. Distribuido bajo los términos de la licencia GNU LGPL ha ganado popularidad como herramienta de soporte a la capa de acceso a datos en el desarrollo de aplicaciones empresariales. Provee mapeo objeto-relacional básico, y otras características sofisticadas como caché y caché distribuida.

Como todas las herramientas ORM, Hibernate busca solucionar el problema de la diferencia entre dos modelos ampliamente utilizados para organizar y manipular datos: el orientado a objetos en las aplicaciones y el relacional en las bases de datos. Para lograr esto el desarrollador debe especificar cómo es su modelo de datos. Con esta información Hibernate permite manipular los datos desde las aplicaciones operando sobre objetos con todas las características de la POO (Programación Orientada a Objetos). Hibernate convierte los datos que define Java a los que define SQL (Structured Query Language), siendo transparente esta conversión para el implementador. Genera además las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias. También ofrece un lenguaje de consulta de datos llamado HQL (Hibernate Query Language) y al mismo tiempo API's (Application Programming Interface) para construir las consultas. (24)

Una de las ventajas de Hibernate es que posee soporte para múltiples dialectos, es decir, puede mapear diferentes sistemas gestores de bases de datos, entre los que se encuentra Postgre, actualmente unos de los más usados por las múltiples prestaciones que ofrece.

1.9 Tecnología para crear Reportes

Es muy importante que los sistemas de gestión de información puedan generar reportes a partir de los datos que perduran en la base de datos, ya que de esta manera los usuarios pueden hacer un análisis más sofisticado y práctico de la información incluso en el momento de tomar decisiones. La generación de reportes sin dudas ayuda a disponer de la información de una manera más rápida y cuenta con medios más flexibles para distribuirla.

¿Qué es un reporte?

Un reporte o informe es una manera organizada de mostrar información procedente de una fuente de datos.

iText es una librería que te permite generar ficheros PDF. A través de éste se puede editar y modificar las propiedades y las características de dichos archivos PDF y además te permite automatizar todos los procesos desde tus propias aplicaciones.

Con iText podrás elegir entre distintas opciones, como servir un PDF a un navegador, generar documentos dinámicos desde ficheros XML o desde bases de datos, añadir bookmarks, poner numeración a las páginas del PDF, aplicar marcas de agua, etcétera. También se logra editar el PDF cortando, concatenando y manipulando diferentes páginas del formato entre sí. (25)

1.10 Gestor de Base de Datos

Es un Sistema de Gestión de Bases de Datos Objeto-Relacional (ORDBMS) basado en el proyecto POSTGRES, de la universidad de Berkeley.

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

1.10.1 PostgreSQL 8.4.6

Es un sistema de base de datos relacional perteneciente al ámbito del software libre que se destaca por su robustez, escalabilidad y cumplimiento de los estándares SQL. Cuenta con versiones para una amplia gama de sistemas operativos, entre ellos: Linux, Windows, Mac SX, Solaris y otros más.

Características de PostgreSQL

- Soporta distintos tipos de datos.
- Incorpora una estructura de datos “array”.

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

1.10.2 MySQL

MySQL es un sistema de gestión de base de datos relacional. Maniobra en una arquitectura cliente/servidor de tal manera que el servidor solo tiene que enviarle una cadena de caracteres y esperar la devolución de los datos. Ha pasado de ser una pequeña base de datos a una completa herramienta. Es una tecnología útil para la creación de bases de datos seguras al poseer un sistema de privilegios y contraseñas que es muy flexible y seguro, que permite verificación basada en el "host".

Las principales características de este gestor de bases de datos son las siguientes:

- Soporta gran cantidad de tipos de datos para las columnas.
- Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP).
- Es relativamente sencillo de añadir otro sistema de almacenamiento. Es útil si desea añadir una interfaz SQL para una base de datos propia.
- Posee gran portabilidad entre sistemas.

Desventajas de MySQL:

- Carece de soporte para transacciones y subconsultas.
- El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación, sobre todo para aquellos programadores que provienen de otros gestores que sí poseen esta característica.
- No es viable para su uso con grandes bases de datos a las que se acceda continuamente, ya que

no implementa una buena escalabilidad.

Gestor de base de datos seleccionado

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

1.11 Servidor Web

Un servidor web es un programa que se ejecuta continuamente en un computador, manteniéndose a la espera de peticiones de ejecución que le hará un cliente o un usuario de Internet. Dentro de los servidores web que se estudiaron en el presente trabajo se tiene a:

- Apache Tomcat
- Roxen

1.11.1 Apache Tomcat

Apache Tomcat es una aplicación de software de código abierto de Java Servlet y tecnologías Java Server Pages. Es desarrollado en un entorno abierto y participativo y publicado bajo la licencia Apache versión 2. Como fue desplegado usando el lenguaje de programación Java, este puede funcionar en cualquier sistema operativo que disponga de la máquina virtual de dicho lenguaje de programación. (26)

Ventajas de Apache Tomcat:

- Tomcat es gratis, es fácil de instalar, se ejecuta en máquinas más pequeñas y es compatible con las API más recientes de Java.

- Su extensibilidad y construcción modular: se pueden poner módulos para ampliar su funcionalidad.
- Robusto y estable.
- Ocupa muy poco espacio, teniendo su código binario (todo clases de Java) un tamaño total de apenas un megabyte, de modo que no es raro que se ejecute tan deprisa.

Desventajas de Apache Tomcat:

- Sólo puede trabajar con un único servidor; no puede utilizarse en clusters.
- Tampoco dispone de un entorno integrado con una sofisticada Graphical User Interface (GUI), asistentes y ayuda en línea.

1.11.2 Roxen

Roxen es un servidor web publicado bajo licencia GNU por un grupo de desarrolladores suecos que posteriormente fundarían la empresa Roxen Internet Services. Desarrollado en el lenguaje de programación Pike, ofrece a los usuarios cientos de módulos que permiten desarrollar fácilmente sitios web muy ricos, dinámicos.

Ventajas de Roxen:

- Multiplataforma, es decir, puede ejecutarse en multitud de plataformas: Windows, Linux, Solaris, MAC OS/X, etc.
- Es de código libre.
- Interfaz de administración vía web muy rica y fácil de usar. (27)

Desventajas de Roxen:

- Dificultad en la seguridad.
- No soporta PHP.
- Permite a cualquier usuario conseguir cualquier archivo de la máquina.

Servidor web seleccionado

Se decide trabajar con Apache Tomcat porque además de las ventajas vistas anteriormente, también se tiene en cuenta que es un servidor web muy fiable, utilizado por innumerables empresas. No se utiliza otros servidores web diferentes del apache, porque éste es el servidor web de más amplio uso en el mundo, con características técnicas como son: soporte de los mejores lenguajes de programación web como PHP, Perl, CGI, Java, Flash, conectividad con bases de datos MySQL, Postgres, Oracle, y administración remota; particularidades que lo hacen la plataforma más robusta y segura del mercado.

1.12 Conclusiones parciales del capítulo

Luego de haber realizado un estudio sobre las tecnologías y herramientas que cumplieran con las tendencias actuales de la programación web y sobre todo orientada al código abierto, se definió como lenguaje de programación Java. Además, fueron seleccionados los framework Spring, Spring-Security, Hibernate, Ext JS y como IDE de desarrollo NetBeans. También se definió RUP como la metodología a utilizar, Visual Paradigm como herramienta CASE para el modelado, como gestor de base de datos PostgreSQL y por último iText para la generación de reportes; todas estas tecnologías con el fin de crear un sistema que cumpla con la calidad requerida.

2 CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

En este capítulo se describe la propuesta de solución del sistema según el problema descrito. Para ello se muestran los procesos del negocio mediante el modelo del dominio. Además, se brinda una concepción general del sistema propuesto y se particulariza en las funcionalidades y características que tendrá el sistema a desarrollar. Se presenta el diagrama de casos de uso del sistema con las correspondientes especificaciones de los casos de uso asociados al sistema. Se describen los patrones de caso de usos empleados.

2.2 Breve descripción del sistema

La herramienta que se propone desarrollar permitirá realizar el proceso de trazabilidad bidireccional de requisitos de una mejor forma, ya que a diferencia de las herramientas que existen ésta podrá utilizarse desde diferentes estaciones de trabajo, permitiendo la conectividad, dado se desarrolla en un entorno Web.

Siguiendo las políticas de seguridad de la universidad, el sistema contará con una previa autenticación de usuarios así como una debida gestión de estos. Según los roles asignados a dichos usuarios así serán los permisos con los que ellos podrán acceder al sistema para así interactuar con las diferentes funcionalidades.

Esta herramienta gestionará todos los requisitos de acuerdo a los flujos de trabajo que se realizan para la construcción de un determinado producto de software. Permitirá al usuario llevar una trazabilidad de las dependencias entre los requisitos, ya sean del cliente o requisitos del producto, visualizándola a través de una matriz de trazabilidad. Brindará la posibilidad de gestionar los diferentes proyectos creados así como cada glosario de términos asociado a dichos proyectos.

2.3 Modelo de Dominio

El modelo de dominio es una representación visual de los conceptos del mundo real significativos para un problema. El diagrama de conceptos del modelo de dominio del presente trabajo se muestra en la Fig.2.1.

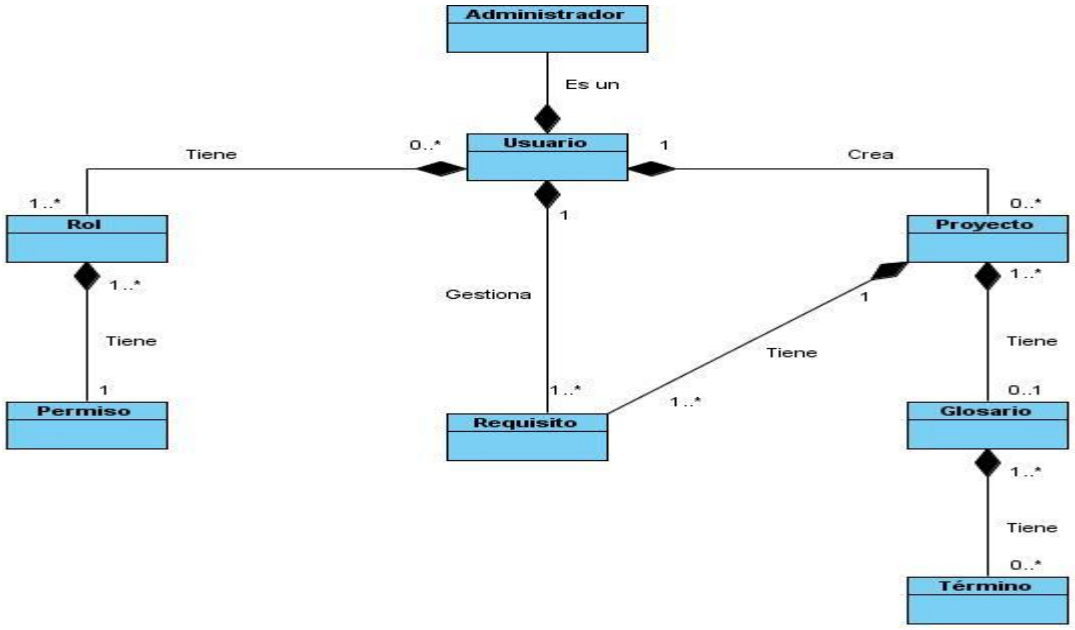

Figura 3: Diagrama de conceptos del modelo de dominio

Para una mejor comprensión, a continuación se describe cada una de las clases que intervienen en el diagrama mostrado:

- **Administrador:** Representa la entidad que dado los privilegios administrativos sobre el sistema, gestionará los usuarios, los roles y la configuración del mismo.
- **Usuario:** Es la entidad que se encarga de gestionar los requisitos y darles seguimiento dentro del sistema.
- **Rol:** Es la entidad que permite que los usuarios o actores tengan un calificativo de acuerdo a la especialidad que desempeñan dentro de un grupo de desarrollo.
- **Proyecto:** Entidad u organización a la que se le van a gestionar los requisitos.

- **Permiso:** Representa la entidad encargada de otorgar ciertos permisos a los roles sobre el sistema.
- **Requisito:** Representa todos los requerimientos (artefactos, requerimientos funcionales y no funcionales) del sistema**.**
- **Glosario:** Representa la entidad que contiene todos los términos asociados al dominio del proyecto en desarrollo.
- **Término:** Representa palabras, siglas etc., que sean de difícil comprensión en el grupo de desarrollo.

2.4 Especificación de los requisitos del sistema

2.4.1 Requisitos Funcionales (RF)

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, y se mantienen invariables sin importar con qué propiedades o cualidades se relacionen. (28)

El sistema a desarrollar a través de la realización del presente trabajo debe cumplir los siguientes requisitos, éstos están agrupados por casos de usos:

RF1- Autenticar Usuario

RF2- Gestionar Usuarios

2.1 Insertar usuario
2.2 Editar usuario
2.3 Eliminar usuario

RF3- Gestionar Requisitos

3.1 Crear requisito
3.2 Editar requisito
3.3 Eliminar requisito

RF4- Gestionar Roles

4.1 Insertar rol

4.2 Editar rol

4.3 Eliminar rol

RF5- Gestionar Proyectos

5.1 Crear proyecto

5.2 Editar proyecto

5.3 Eliminar proyecto

RF6- Generar Matriz de Trazabilidad

RF7- Gestionar Configuración

7.1 Crear configuración

7.2 Editar configuración

7.3 Eliminar configuración

7.4 Visualizar detalles de la configuración sobre requisitos de proyectos

7.5 Visualizar detalles de la configuración sobre requisitos de flujos de trabajo

7.6 Establecer permisos sobre específicos flujos de trabajo

7.7 Establecer permisos sobre determinados proyectos

RF8- Generar Reportes

8.1 Obtener requisitos creados por un usuario

8.2 Obtener proyectos creados por un usuario

8.3 Obtener requisitos de un determinado flujo de trabajo

8.4 Obtener todos los términos de un glosario

8.5 Obtener los requisitos de un tipo de estado

8.6 Obtener roles de un usuario

8.7 Obtener los cambios realizados a un requisito

RF9- Gestionar Glosario de Términos

9.1 Crear glosario

9.2 Editar glosario

9.3 Eliminar glosario

RF10- Gestionar Términos

10.1 Crear término

10.2 Editar término

10.3 Eliminar término

2.4.2 Requisitos No Funcionales (RNF)

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener, y que harán del mismo un sistema confiable y seguro. (28)

RNF 1 – Requerimientos de Apariencia o Interfaz Externa

- El sistema deberá poseer una interfaz web sencilla, amigable, lo más atractiva y clara posible para el usuario.

RNF 2 - Requerimientos de Uso

La aplicación garantiza una fácil interacción entre cliente y PC, de tal forma que no haya conflictos de usabilidad entre ambos. El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora y de un ambiente Web en sentido general.

RNF 3 - Requerimientos de Seguridad

- **Confidencialidad**

La autenticación será la primera acción del usuario en el sistema y consistirá en proveer un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica.

- **Integridad**
  Se debe garantizar que la información sensible sólo pueda ser vista por los usuarios con el nivel de acceso adecuado y que las funcionalidades del sistema se muestren de acuerdo al usuario que esté activo.
- **Disponibilidad**
  El sistema debe estar disponible para su utilización las 24 horas del día, durante los siete días de la semana, con el menor tiempo posible de recuperación ante fallos.

RNF 4 - Requerimientos de software

Cliente

- Debe tener instalado un navegador web que soporte JavaScript y Adobe Reader 5 o superior. Se recomienda Mozilla Firefox 3.5 o superior, Google Chrome 3.0 o superior, Internet Explorer 6 o superior, Opera 9 o superior o Safari 3 o superior.
- Debe tener instalado Sistema Operativo: Windows XP o superior, o GNU Linux.

Servidor

- Se debe instalar TOMCAT como servidor web y PostgreSQL como gestor de base de datos.
- Debe estar instalado el Java Runtime Environment (JRE) versión 1.6 o superior.
- Debe estar instalado Sistema Operativo: Windows XP o superior, o GNU Linux.

RNF 5 - Requerimientos de Hardware

- Para las PC clientes: procesador Pentium II o superior, 256 Mb de RAM.
- El servidor de aplicaciones debe tener las siguientes características: capacidad de disco duro superior a 80 GB, microprocesador Pentium IV superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

- El servidor de base de datos debe tener las siguientes características: capacidad de disco duro superior a 180 GB, microprocesador Pentium IV superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

RNF 6- Requerimientos de restricciones en el diseño y la implementación

- Los componentes del sistema deben desarrollarse siguiendo el principio de alta cohesión y bajo acoplamiento.
- La lógica de presentación constituirá una capa independiente de la lógica de negocio, centrando su función en la interfaz de usuario y validaciones de los datos de entrada.

2.5 Definición de los casos de usos del sistema

2.5.1 Actores del sistema

Los actores representan a terceros fuera del sistema que interactúan con él. En el sistema que se describe se identificaron los siguientes actores:

Actor	Descripción
Administrador	Es una generalización del usuario del sistema y es el encargado de gestionar los usuarios, los roles que van a tener dichos usuarios así como la configuración de los permisos para acceder a las determinadas funcionalidades.
Usuario	Rol que representa a los usuarios del sistema que se han autenticado y pueden acceder a los recursos que le

	son permitidos.
Gestor de Proyectos	Este rol constituye una generalización del usuario del sistema y se ocupa de gestionar los proyectos.
Gestor de Reportes	Es una generalización del usuario del sistema, es el encargado de la gestión de los reportes.
Gestor de Requisitos	Representa una generalización del usuario del sistema, se encarga de gestionar los requisitos.
Gestor de Trazabilidad	Constituye una generalización del usuario del sistema es el encargado de generar la matriz de trazabilidad entre los requisitos.
Gestor de Glosarios y Términos	Representa una generalización del usuario del sistema y gestiona los glosarios y términos incluidos en el dominio de cada proyecto involucrado.

Tabla 1: Descripción de los actores del sistema.

2.5.2 Lista de los casos de uso del sistema

La forma en que los actores usan el sistema es representada a través de los casos de usos. Estos últimos son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del actor. Los casos de usos identificados en el presente trabajo son enunciados a continuación:

Orden	Nombre	Prioridad	Breve descripción
1	Autenticar Usuario	Crítico	Inicia el caso de uso cuando un usuario decide interactuar con el sistema. El sistema realiza las verificaciones de los datos enviados y finalmente lo acepta o lo rechaza.
2	Gestionar Usuarios	Crítico	El caso de uso inicia cuando el actor envía la petición para insertar, modificar, buscar y visualizar o eliminar usuarios. Una vez realizada alguna de estas acciones finaliza el caso de uso.
3	Gestionar Requisitos	Crítico	El caso de uso inicia cuando el actor envía la petición para insertar, modificar, buscar y visualizar o eliminar requisitos. Una vez realizada alguna de estas acciones finaliza el caso de uso.
4	Gestionar Roles	Secundario	El caso de uso inicia cuando el actor envía la petición para crear, modificar, buscar y visualizar o eliminar roles. Una vez realizada alguna de estas acciones finaliza el caso de uso.

5	Gestionar Proyectos	Secundario	El caso de uso inicia cuando el actor envía la petición para crear, cargar, buscar y visualizar o eliminar proyectos. Una vez realizada alguna de estas acciones finaliza el caso de uso.
6	Generar matriz de Trazabilidad	Crítico	El caso de uso inicia cuando el actor envía la petición generar la matriz de trazabilidad entre los requisitos que seleccione. Una vez realizada esta acción finaliza el caso de uso.
7	Gestionar Configuración	Crítico	El caso de uso inicia cuando el actor envía la petición para crear, modificar, buscar y visualizar o eliminar configuración. Una vez realizada alguna de estas acciones finaliza el caso de uso.
8	Gestionar Reportes	Secundario	El caso de uso inicia cuando el actor envía la petición para crear, modificar, buscar y visualizar o eliminar reportes. Una vez realizada alguna de estas acciones finaliza el caso de uso.
9	Gestionar Glosario de Términos	Secundario	El caso de uso inicia cuando el actor envía la petición para crear, modificar, buscar y visualizar o eliminar glosario de términos. Una vez realizada

			alguna de estas acciones finaliza el caso de uso.
10	Gestionar Términos	Secundario	El caso de uso inicia cuando el actor envía la petición para crear, modificar, buscar y visualizar o eliminar términos de un glosario de términos. Una vez realizada alguna de estas acciones finaliza el caso de uso.

Tabla 2: Breve descripción de los casos de usos del sistema.

2.5.3 Diagrama de casos de uso del sistema

Los diagramas de casos de uso del sistema representan gráficamente a los procesos y su interacción con los actores.

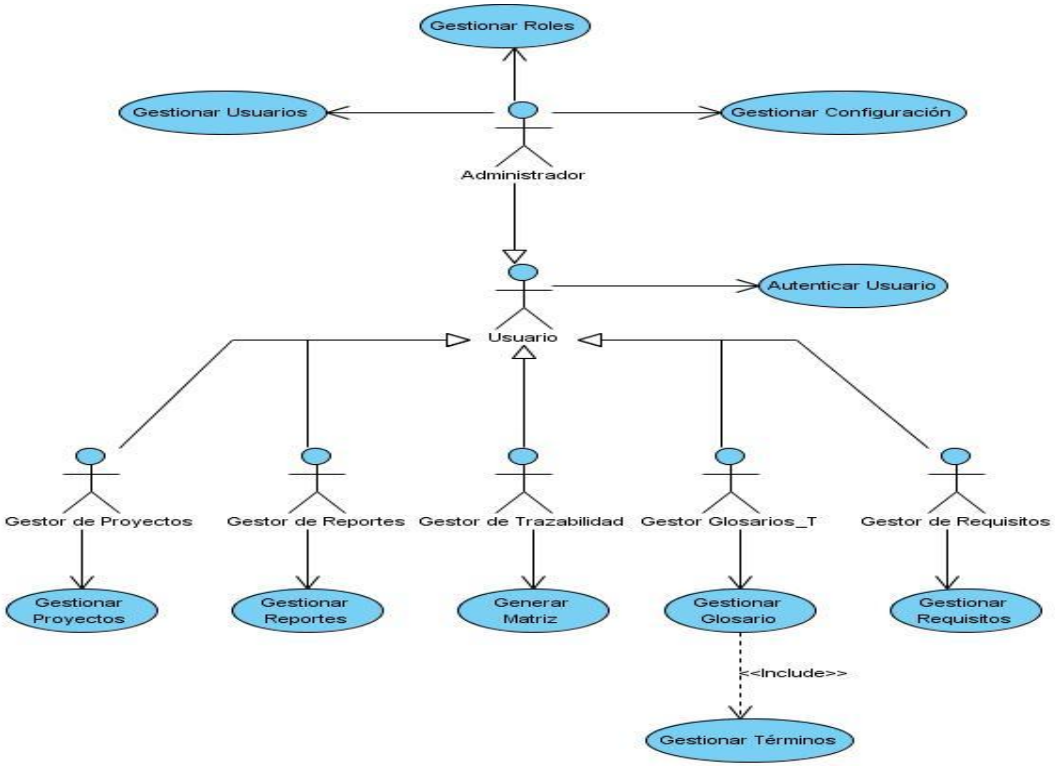

Figura 4: Diagrama de Casos de Uso del Sistema

2.6 Especificación de los casos de uso del sistema

A continuación se muestra la descripción del caso de uso Gestionar Requisitos, éste es un caso de uso crítico dentro del sistema. Las descripciones de los restantes casos de uso formarán parte del expediente de proyecto.

Caso de Uso:	Gestionar Requisitos
Actores:	Gestor de Requisitos
Propósito:	Gestionar toda la información referente a los requisitos.

Resumen:	El caso de uso se inicia cuando el usuario decide gestionar los requisitos, el sistema brinda la posibilidad de crear un nuevo requisito, editar o eliminar.
Referencia:	RF3.1, RF3.2, RF3.3
Precondiciones:	El usuario debe estar autenticado con el rol correspondiente
Poscondiciones:	Queda actualizada la lista de requisitos.

Tabla 3: Descripción del CU Gestionar Requisitos.

2.7 Patrones de casos de uso utilizados

¿Qué es un patrón?

Pareja de **problema / solución** con un nombre, que codifica (estandariza) buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades.

- **Múltiples actores**

Roles comunes:

Puede suceder que los dos actores jueguen el mismo rol sobre el caso de uso. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del caso de uso, sólo exista una entidad externa interactuando con cada una de las instancias del caso de uso.

A continuación se presenta un ejemplo donde se evidencia dicho patrón. En este caso los actores “Administrador” y “Gestor de Proyecto”, ambos son usuarios del sistema por lo que para acceder al mismo deben ejecutar el caso de uso “Autenticar Usuario”.

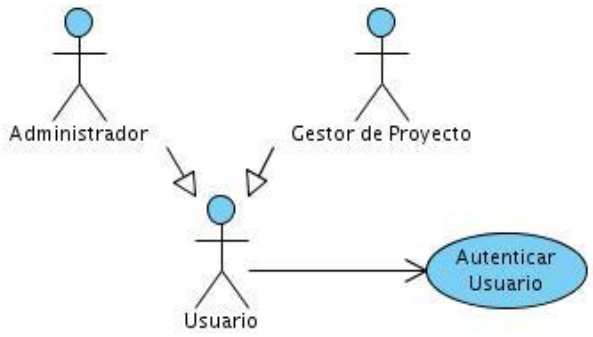

Figura 5: Ejemplo del patrón múltiples actores, roles comunes

- **CRUD (Creating, Reading, Updating, Deleting)**

Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual.
Este patrón consta de un caso de uso, llamado Información CRUD o Gestionar información, modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y éstos a su vez son cortos y simples.

A continuación se presenta un ejemplo donde se evidencia dicho patrón. En este caso existe un actor llamado “Administrador” que ejecuta el caso de uso “Gestionar Roles”. Este caso de uso agrupa un conjunto de funcionalidades asociadas a un mismo tipo de información, como: insertar, modificar, buscar y visualizar, eliminar roles.

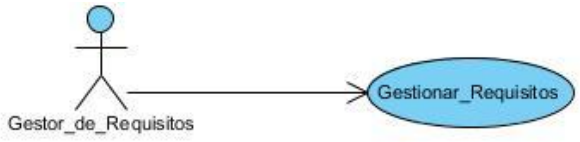

Figura 6: Ejemplo del Patrón CRUD

2.8 Conclusiones parciales del capítulo

En este capítulo se identificaron 34 requisitos funcionales, que agrupados por patrones de casos de uso, forman 10 casos de uso del sistema, de ellos 5 son críticos. Se determinaron además los requisitos no funcionales, destacando los de seguridad, software y hardware, así como sus requerimientos mínimos para un funcionamiento adecuado, por último se evidenciaron los patrones de caso de uso utilizados. Tomando como base los casos de uso del sistema, y la vista de casos de uso arquitectónicamente significativos se puede comenzar el diseño del producto que se presenta en este trabajo.

3 CAPÍTULO 3: DISEÑO DEL SISTEMA

3.1 Introducción

En este capítulo se traducen los requisitos a una especificación que describe cómo implementar el sistema a través del diseño, enfocado a cómo el sistema cumple sus objetivos teniendo en cuenta los requisitos funcionales y no funcionales, se realizan los diagramas de clases y los diagramas de secuencias más relevantes según los casos de usos definidos en el capítulo anterior. También se muestra el modelo entidad-relación (MER) y se explica además la arquitectura y los principales patrones de diseño utilizados.

3.2 Estilo Arquitectónico Utilizado

El Estilo Arquitectónico es uno de los aspectos fundamentales de la disciplina Arquitectura de Software. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. (29)

¿Qué es un patrón de arquitectura?

Expresan un paradigma fundamental para estructurar u organizar un sistema software. Proporciona un conjunto de subsistemas o módulos predefinidos, con reglas y guías para organizar las relaciones entre ellos. (28)

Para la realización del sistema el colectivo de autores decidió utilizar el estilo arquitectónico: Modelo Vista Controlador (MVC), perteneciente a la familia de estilos arquitectónicos de Llamada y Retorno, basándose en las características y peculiaridades de dicho sistema.

El estilo arquitectónico MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Para ello utiliza tres clases diferentes: Modelo, Vista y Controlador. (30)

- **Modelo**: Esta es la representación específica de la información con la cual el sistema opera. Se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- **Vista:** Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- **Controlador:** Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

Este estilo tiene soporte de vistas múltiples, dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Además, posee adaptación al cambio. (29)

3.3 Patrones de diseños utilizados

Inyección de dependencias: es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. Es una forma de Inversión de Control, que está basada en constructores de Java. Este radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todos los usuarios (al menos en el caso de Spring) y evitando tener que andar extendiendo clases.
En el sistema está presente su aplicación ya que a todos los controladores se le inyectan las fachadas correspondientes para su funcionamiento mediante el uso de la anotación @Resource la cual provee de atributos a clases mediante los métodos set, de esta misma manera se soluciona la dependencia entre las fachadas y sus servicios, dependientes estos últimos del DAO Genérico.

Data Access Object (DAO): Patrón de Diseño Core J2EE que centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Sus principales beneficios son que reduce la

complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos y que permite una migración más fácil de fuente de datos.

El uso de este patrón en el sistema se evidencia en la capa de Acceso a Datos donde se encuentra un DAO Genérico encargado de realizar la gestión de los datos entre las clases que contienen la lógica de negocio (servicios) y el ORM Hibernate.

Patrones GRASP: Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. (31)
Ejemplo de los patrones GRASP que se evidencian en la aplicación se tiene al patrón Bajo Acoplamiento en la clase UserAppJpaController.java que éste funciona cuando se provoca la asignación de responsabilidad de modo que su colocación no incremente el acoplamiento generando resultados negativos propios de un alto ajuste.
También se pone de manifiesto el patrón Controlador en la clase UserAppController.java el cual se utiliza para asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.)
Y otro patrón que se afirma es el Creador en la clase Requirements.java. La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En síntesis es útil contar con un principio general para la asignación de las responsabilidades de creación y si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

3.4 Diagramas de Clases del Diseño

Los diagramas de clases son ampliamente utilizados en el modelado de sistemas orientados a objetos, empleándose para representar las relaciones que se establecen entre las clases. Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces de la aplicación. Sirve también para visualizar las relaciones entre las clases que involucran el sistema.

A continuación se muestra en la (Fig. 3.1) el diagrama de clases del diseño para el caso de uso Gestionar Requisitos.

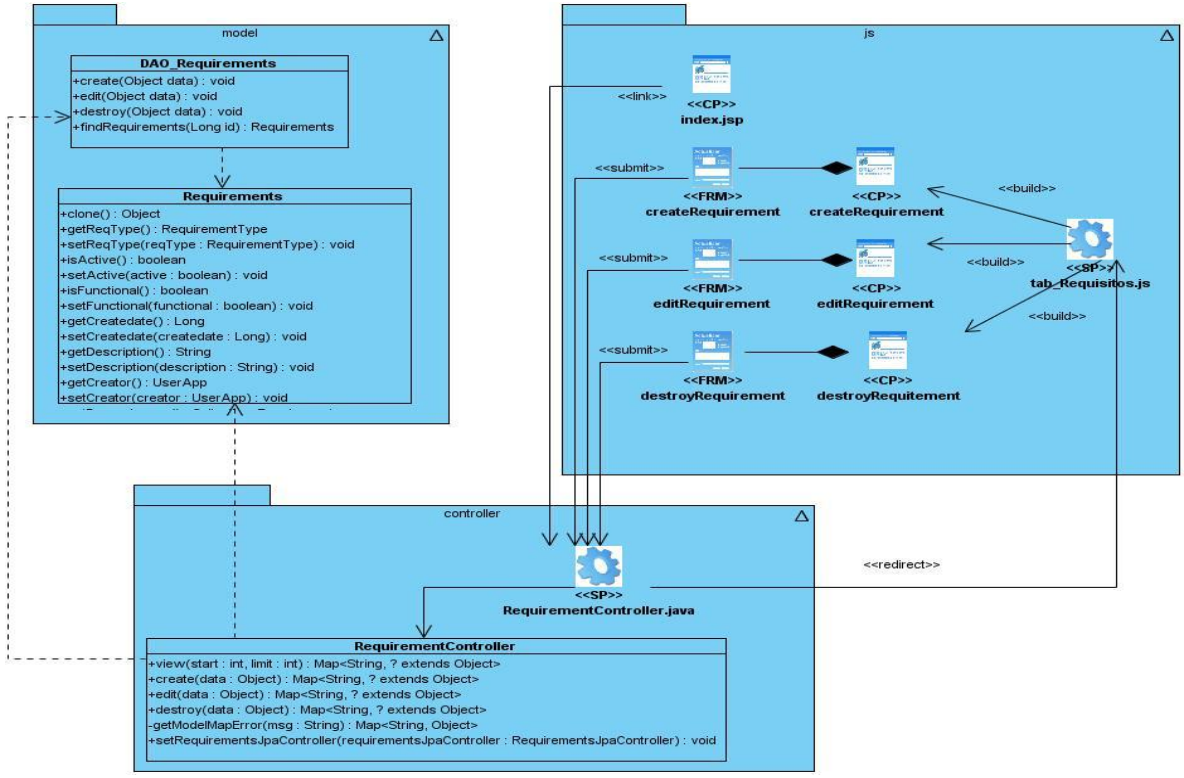

Figura 7: Diagrama de Clases del Diseño del Caso de Uso Gestionar Requisitos

3.5 Diagramas de Secuencia

Para la realización de los casos de uso del diseño es más factible el empleo de los diagramas de secuencia ya que representan con más claridad el flujo de las acciones que debe realizar el sistema.

Un diagrama de secuencia muestra las interacciones entre objetos ordenados en secuencia temporal; muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos, para llevar a cabo la funcionalidad descrita por el escenario.

A continuación se muestran los diagramas de secuencia correspondientes a los escenarios:

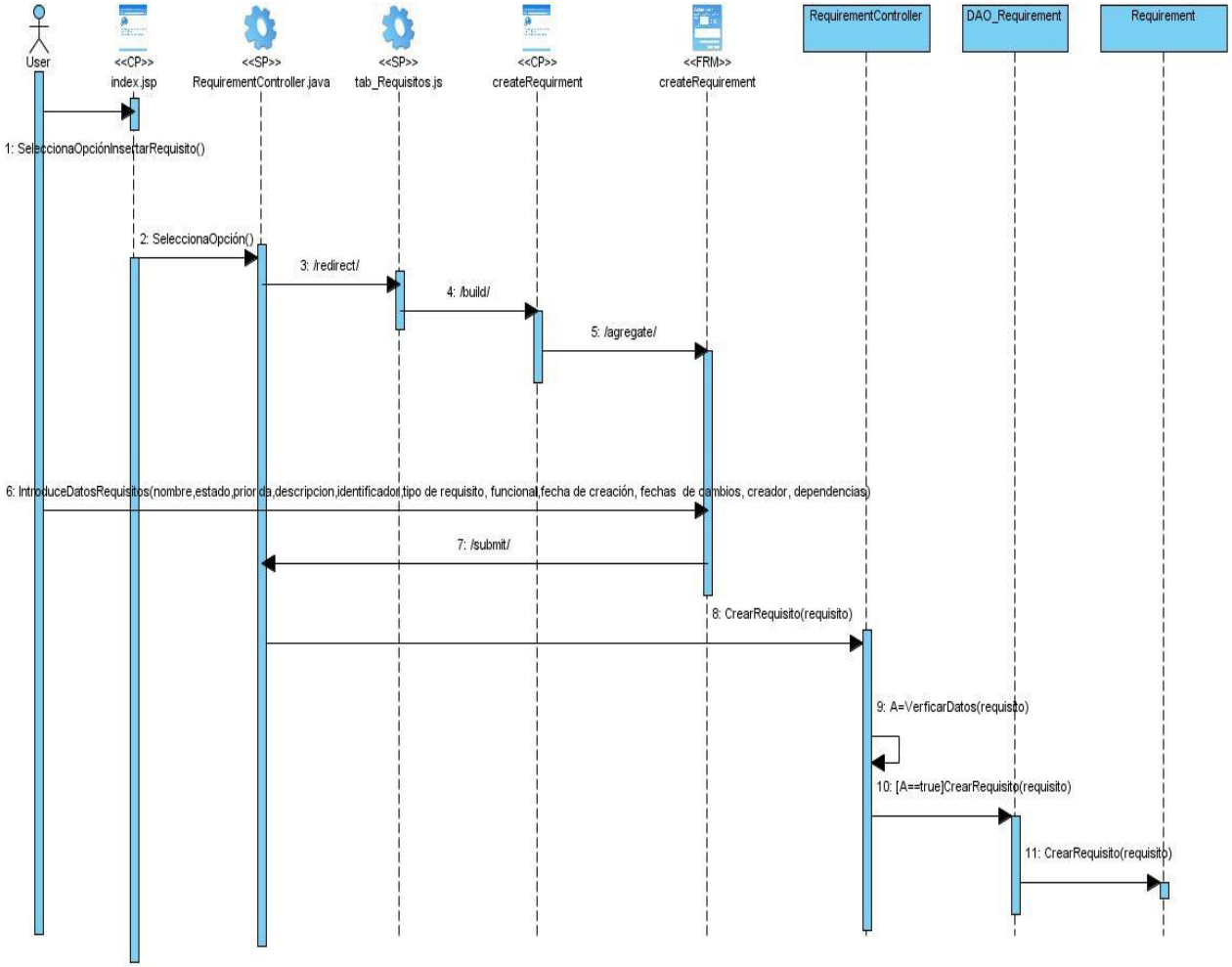

Figura 8: Diagrama de Secuencia del Caso de Uso Gestionar Requisitos Escenario Crear Requisito

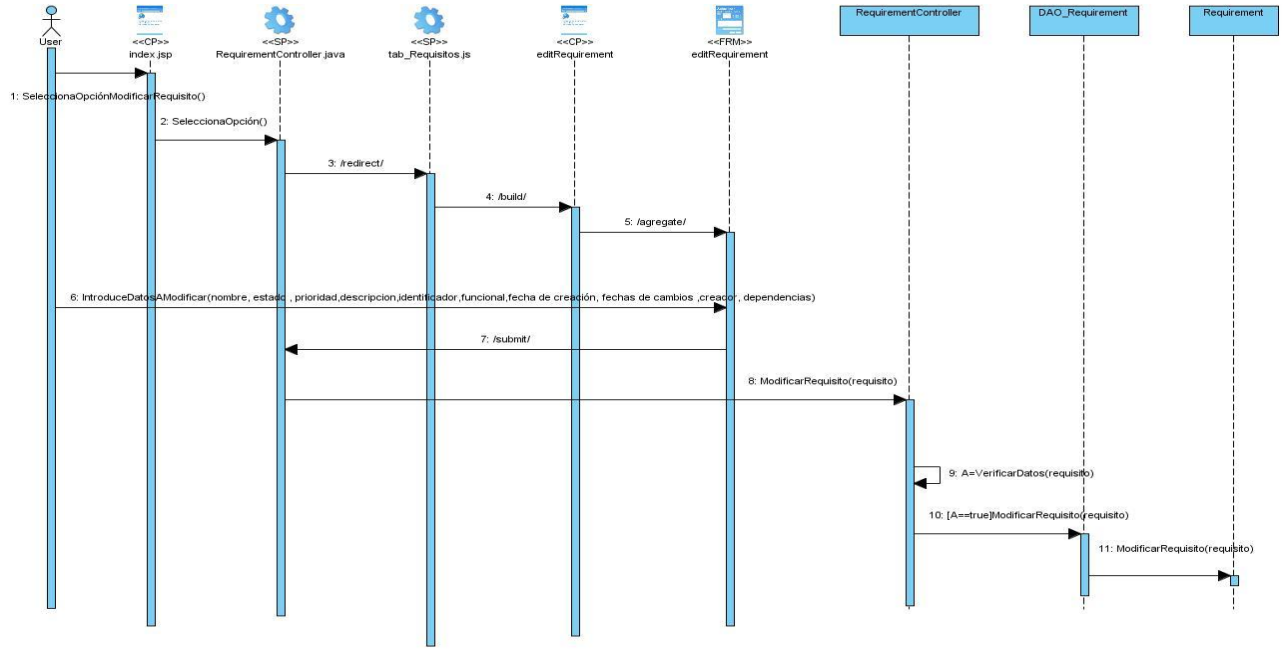

Figura 9: Diagrama de Secuencia del Caso de Uso Gestionar Requisitos Escenario Modificar Requisito

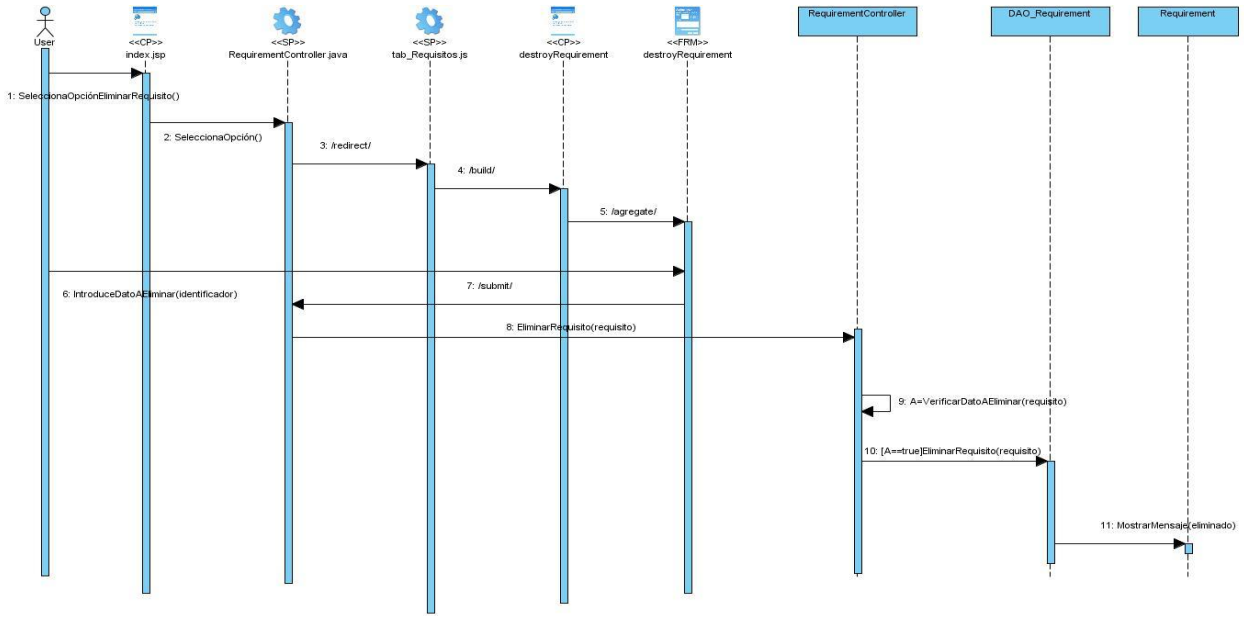

Figura 10: Diagrama de Secuencia del Caso de Uso Gestionar Requisitos Escenario Eliminar Requisito

3.6 Diagramas de Clases Persistentes

La persistencia es la propiedad de los datos para subsistir de una manera u otra. En lo general las clases persistentes tienen como origen las clases clasificadas como entidad porque ellas modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto del mundo real o un suceso. (32)

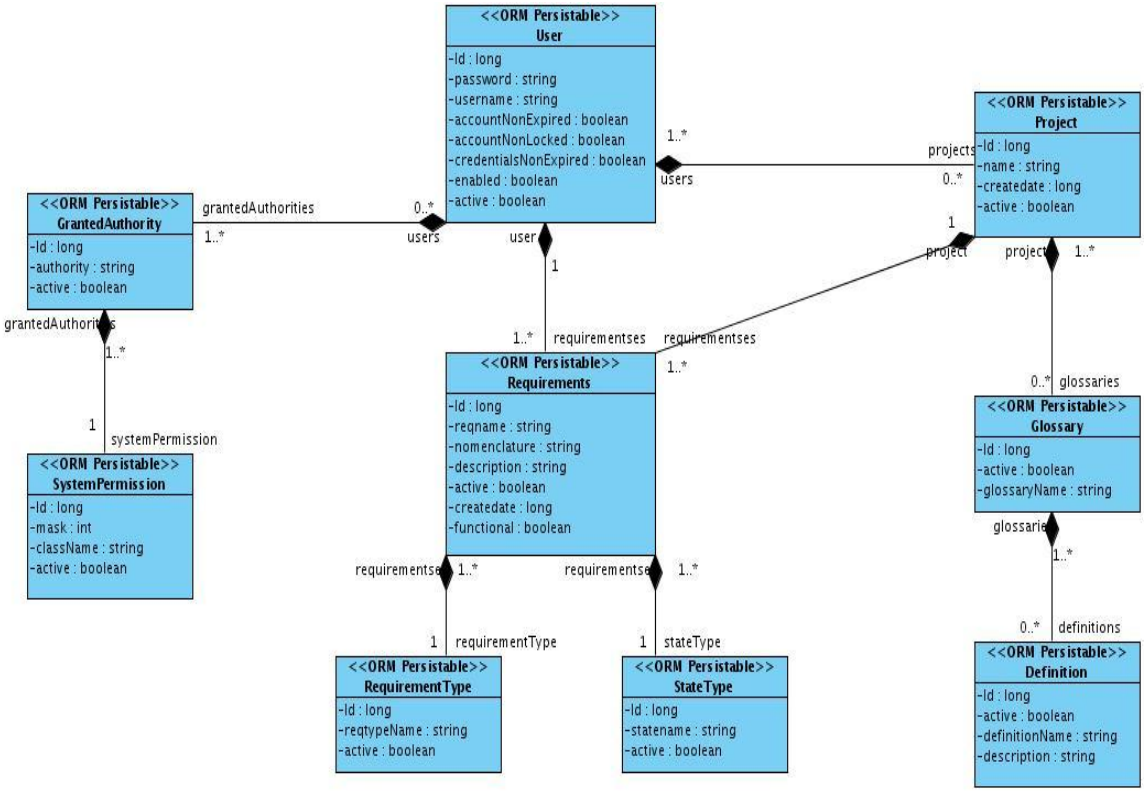

Figura 11: Diagrama de Clases Persistentes

3.7 Modelo de Datos

El Modelo Entidad Relación es un tipo de diagrama para modelado de base de datos. Su objetivo es representar relaciones que existen en la vida real entendiendo su semántica.

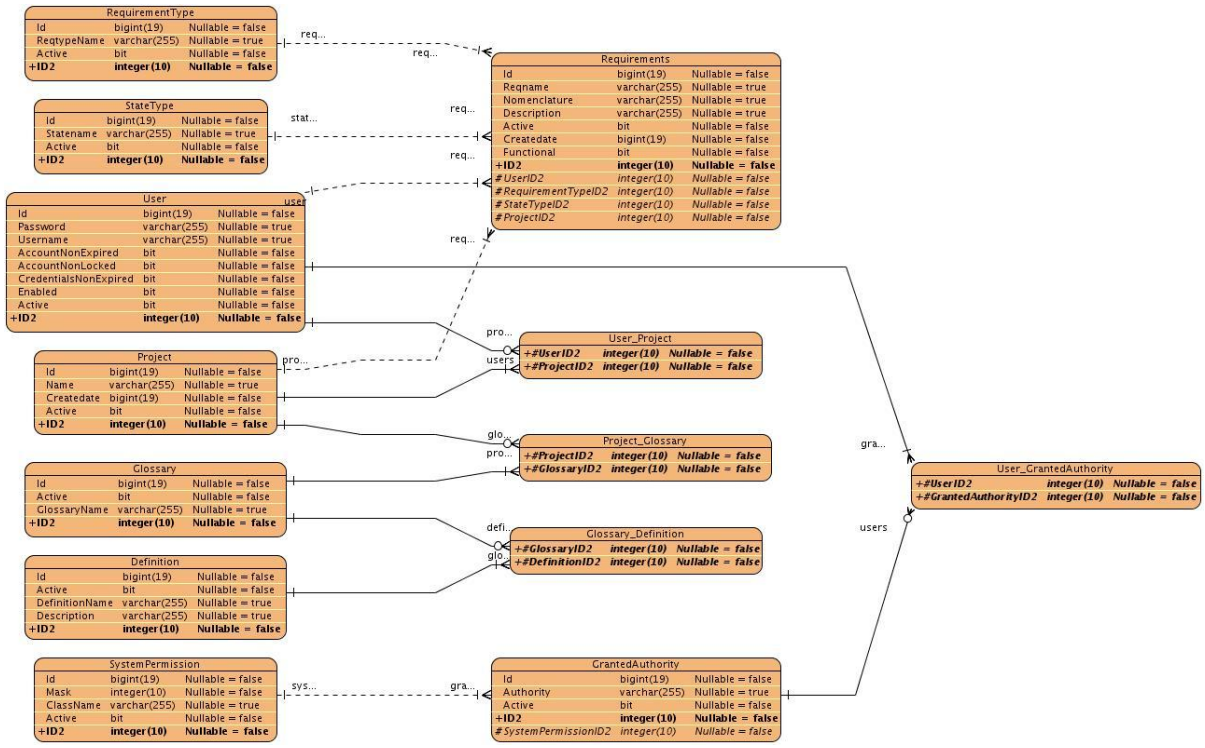

Figura 12: Modelo de Datos

3.8 Modelo de despliegue

El Modelo de Despliegue se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos.

El diagrama de despliegue realizado representa cuatro nodos principales. El nodo PC_Cliente que requiere de un navegador que soporte Java Script y Adobe Reader 5.0 o superior, el nodo Servidor_Web, en el cual debe estar instalado el servidor Apache Tomcat, en el nodo Servidor_BD debe estar instalado el SGBD PostgreSQL .

El nodo PC_Cliente estará conectado mediante el Protocolo de Transferencia de Hipertexto HTTP al nodo procesador que representa al Servidor Web. La conexión entre el Servidor Web y el Servidor de Base de Datos se realizará mediante el protocolo de comunicación TCP/IP.

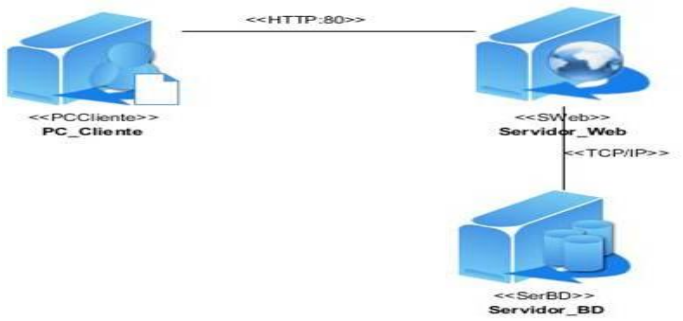

Figura 13: Diagrama de Despliegue

3.9 Conclusiones parciales del capítulo

En el presente capítulo se hizo uso del estilo arquitectónico MVC, debido a que es el patrón de diseño arquitectónico recomendado para aplicaciones interactivas en Java. Se aplicaron varios patrones de diseño, dentro de los más relevantes se tienen inyección de dependencias, el patrón DAO y dentro de los GRASP, en específico, bajo acoplamiento, el controlador y el creador. Se realizaron diagramas de clases del diseño y los diagramas de secuencia de los diferentes escenarios de cada caso de uso.

4 CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

Durante este capítulo se realiza el flujo de trabajo de implementación, se analizan los principales artefactos como el modelo de implementación que incluye componentes, y diagramas de éstos. Se comienza a implementar el sistema en términos de componentes mediante las clases del diseño. Además una vez finalizado el software se realizarán las pruebas.

4.2 Modelo de implementación

Describe cómo los elementos del modelo de diseño se implementan en términos de componentes y también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación.
Es una colección de componentes y subsistemas de la aplicación que los contengan. Los componentes incluyen tanto los elementos a implementar, tales como ejecutables, y los componentes de los productos que se producen, tales como archivos de código fuente.

4.2.1 Diagrama de componentes

Dentro del modelo de implementación se encuentran los diagramas de componentes. Un diagrama de componentes muestra las dependencias lógicas entre componentes software, sean estos elementos fuentes, binarios o ejecutables; ilustran las piezas del software y controladores embebidos. Los diagramas de componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. (33)En la figura siguiente se muestra el diagrama de componentes para el caso de uso “Gestionar requisitos”.

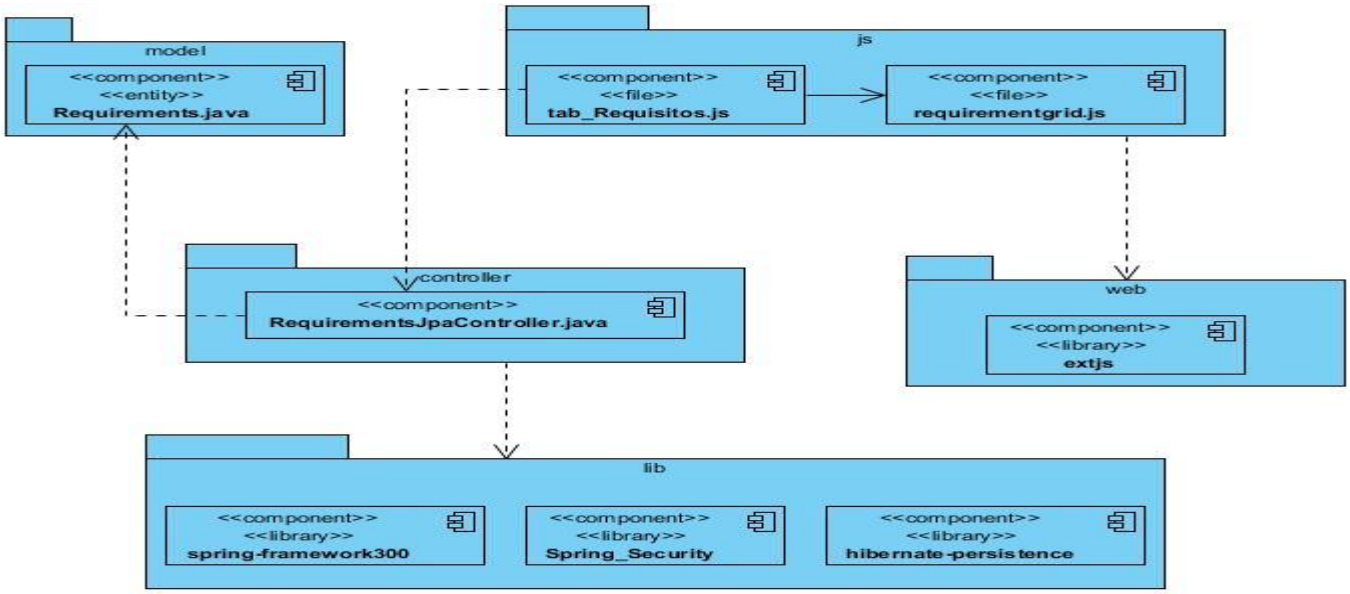

Figura 14: Diagrama de componente del caso de uso Gestionar Requisitos

4.3 Código fuente

Luego de implementar los componentes que se han definido se obtiene un conjunto de instrucciones que componen el programa informático, es a lo que se le denomina código fuente.

También se nombra fuente o texto fuente que contiene las instrucciones del programa, escritas en un lenguaje de programación. Se trata de un archivo de texto legible que se puede copiar, modificar e imprimir sin dificultad. (34)

4.3.1 Estándares de codificación

Las convenciones o estándares de codificación son un conjunto de directrices que especifican cómo debe escribirse el código fuente. (35)

Declaraciones de clases

Se siguen estas reglas:

- No hay espacio entre el nombre del método, el paréntesis y la lista de parámetros.
- Se abre la llave {al final de la misma línea que la declaración.

- La llave de} debe aparecer en línea aparte con la misma indentación que el método o clase que cierra.

Asignación de nombres

- Se usan descriptores en inglés que dejan claro el cometido de la variable, método o clase.
- Se usa terminología aplicable al dominio.
- Si se usan abreviaturas hay que mantener en algún sitio una lista de lo que significan.

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

- Paquetes: Todo en minúsculas. Cada palabra es más específica que la anterior
- Clases e interfaces: Nombres. La inicial en mayúscula.
- Métodos: Deben ser verbos. La primera letra de la primera palabra en minúsculas.

4.3.2 Ejemplo de código fuente

La mayoría de los métodos implementados responden a funciones básicas de inserción, actualización, y eliminación generando de esta forma la mayor parte del código de acceso a datos. A continuación se muestra en la figura el código del método Crear Requisito y se ofrece una breve descripción de este.

```
public void create(Object data) throws AccessForbiddenException
{
    if(!GrantsChecker.checkGrants(4L, "ToRequirementType"))
       throw new AccessForbiddenException("No tiene permiso para crear tipos de requisitos"
    else{
        EntityManager em = null;

        RequirementType requirementType;
        try
        {
            requirementType = new Gson().fromJson((String) data, RequirementType.class);
        }
        catch (Exception ex)
        {
            requirementType = new RequirementType();
        }

        try
        {
            em = getEntityManager();
            em.getTransaction().begin();
            em.persist(requirementType);
            em.getTransaction().commit();
        }
        finally
        {
            if (em != null)
            {
                em.close();
            }
        }
```

Figura 15: Ejemplo de código fuente: Método Crear Requisito

4.4 Pruebas de software

Uno de los principales problemas de los proyectos de desarrollo de software ha sido garantizar la calidad del producto final. Esta es dependiente de la calidad del proceso que se siga, por lo que si se lleva a cabo un proceso defectuoso, las probabilidades de que el producto también lo sea son altas. En el flujo de trabajo de pruebas es donde se va a verificar el resultado de la implementación. Ellas no son más que un conjunto de herramientas, técnicas y métodos que garantizan el buen desempeño de una aplicación para la validación del producto.

4.4.1 Tipos de pruebas

Las pruebas de software involucran las operaciones del sistema bajo condiciones controladas y evaluando los resultados.

Niveles de Prueba:

- Pruebas unitarias:

Se enfocan en un programa o un componente que desempeña una función específica que puede ser probada y que se asegura que funcione tal y como lo define la especificación del programa. Los programadores siempre prueban el código durante el desarrollo, por lo que las pruebas unitarias son realizadas solamente después de que el programador considera que el componente está libre de errores. (36)

- Pruebas de integración:

Se comprueba la compatibilidad y funcionalidad de los interfaces entre las distintas 'partes' que componen un sistema, estas 'partes' pueden ser módulos, y aplicaciones cliente/servidor.

- Pruebas de sistema:

Por los general se realizan sobre el sistema funcionando y se comprueba que cumplen con la especificación (normalmente a través de los casos de uso). Concretamente se debe comprobar que:

Se cumplen los requisitos funcionales establecidos.

- ✓ Sea correcto el funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
- ✓ Sea apropiada la documentación de usuario.
- ✓ Se verifique el rendimiento y respuesta en condiciones límite y de sobrecarga.

- Pruebas de aceptación:

El cliente comprueba que el software funciona según sus expectativas. (36)

Métodos de prueba:

- Pruebas de caja blanca:

Este tipo de pruebas se realiza con conocimiento de la estructura interna del programa y el objetivo principal es probar que todos los caminos del código están correctos. Son usadas por lo general en pruebas unitarias y son realizadas por los programadores.

- Pruebas de caja negra:

No es necesario conocer los detalles internos del programa y el objetivo fundamental de estas es probar que también el software está conforme a los requisitos. Este tipo de pruebas son usadas por lo general en pruebas de integración y del sistema y pueden ser realizadas por los usuarios.

4.4.2 Pruebas realizadas

Para garantizar la calidad del software y validar el sistema se realizaron pruebas con el método de caja negra, procedimiento que es capaz de encontrar errores en la interfaz, en las estructuras de datos, en el acceso a la base de datos entre otros.

Como resultado de estás pruebas se realizó una gráfica mostrando la cantidad de no conformidades encontradas por cada caso de uso y el tipo de no conformidad definido de la siguiente manera:

- No conformidades de prioridad baja a las validaciones en los campos de texto, errores en la interfaz y en la representación de los datos en los reportes.
- No conformidades de prioridad media a los errores encontrados relacionados con el establecimiento de la conexión a la base de datos y actualización de los datos en la misma.
- No conformidades de prioridad alta a los errores de funcionalidades que no estén implementadas correctamente.

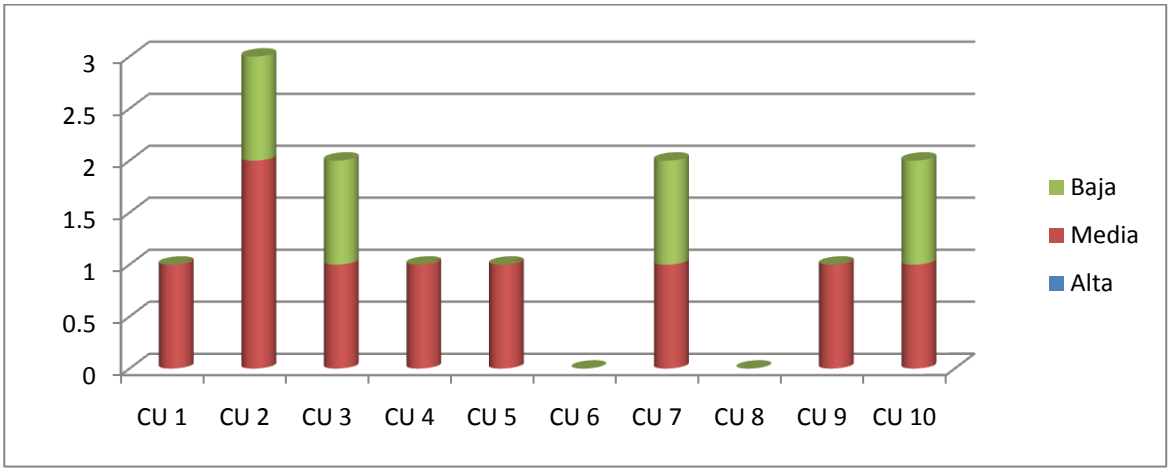

Tabla 4: Gráfico de cantidad de no conformidades por casos de uso.

Los casos de uso que se muestran en el gráfico siguen el orden siguiente:

CU 1- Autenticar Usuario

CU 2- Gestionar Usuarios

CU 3- Gestionar Requisitos

CU 4- Gestionar Roles

CU 5- Gestionar Proyectos

CU 6- Generar Matriz de Trazabilidad

CU 7- Gestionar Configuración

CU 8- Gestionar Reportes

CU 9- Gestionar Glosario de Términos

CU 10- Gestionar Términos

Con la realización de estas pruebas, se pudo no solo identificar sino corregir los errores; de los cuales es importante mencionar que no se encontró ninguno de prioridad alta, esto demuestra la estabilidad de la

solución propuesta. Se efectuaron dos iteraciones para corregir las no conformidades que existían en la aplicación. En el expediente de proyecto se encuentran plasmados todos los casos de prueba correspondientes a esta etapa con los resultados que arrojaron.

4.5 Conclusiones parciales

En el capítulo se detalla la fase de implementación de la solución propuesta en la investigación, así como la realización de las pruebas de caja negra realizadas, con el objetivo de validar la aplicación y que esta cumpla con los requisitos funcionales propuestos. Durante este proceso se detectaron 8 no conformidades que fueron resueltas por los desarrolladores.

CONCLUSIONES

Una vez finalizado el trabajo es posible afirmar que se les dio cumplimiento a los objetivos trazados:

- Se realizó un levantamiento de todas las funcionalidades requeridas para la aplicación y se alcanzó el desarrollo del sistema informático.
- Se diseñaron todos los artefactos relacionados con la implementación del sistema.
- El sistema informático fue implementado utilizando herramientas, lenguajes y tecnologías en su mayoría distribuidas bajo licencias de software libre en correspondencia con el principio de independencia tecnológica en el cual está basado actualmente el desarrollo de software en Cuba.
- Se realizaron pruebas funcionales utilizando el método de caja negra así como diseños de casos de prueba basados en casos de uso para verificar el correcto funcionamiento de la aplicación.

RECOMENDACIONES

- Implementar el sistema de trazabilidad bidireccional basado en ontologías, debido a que el trabajo con ellas permite la representación del conocimiento en un determinado dominio, lo que ayuda en una mejor reutilización, razonamiento, organización y compartición de la información.
- Agregar un módulo estadístico de estimación de proyectos aplicando técnicas que permitan la resolución de problemas de costo y esfuerzo.
- Integrar el sistema informático del presente trabajo de diploma con la herramienta Acaxia y el generador dinámico de reporte.
- Realizar pruebas unitarias y pruebas de rendimiento a la aplicación.

REFERENCIAS BIBLIOGRÁFICAS

1. Carlos Pérez. Suite101.net. *Suite101.net.* [Online] julio 10, 2010. [Cited: noviembre 20, 2010.] http://www.suite101.net/content/modelo-cmmi-a18334...
2. Amador Duran Toro. *Un Entorno Metodológico de Ingenieria de Requisitos para Sistemas de Información.* Sevilla, España : s.n., 2000.
3. IEEE-SA. IEEE Standard Glossary of Software Engineering Terminology. [Online] mayo 24, 2006. [Cited: diciembre 1, 2010.] http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html. 610.12-1990..
4. Patricio Letelier. *Trazabilidad en el Proceso de Desarrollo de Software.* Valencia, España : s.n.
5. Victor Anaya, Patricio Letelier. *SmarTTrace: Una Herramienta para Trazabilidad de Requisitos en Proyectos basados en UML.* Valencia, España : s.n., 2002.
6. Elena Sanchez Pescador. *Analisis e implatacion de una herramienta de gestion de requisitos para la gestion de servicios basado en la filosofia de itil V3, CMMI de servicios y MOF.* Madrid, España : s.n., 2009.
7. ATOS, ITA, TID, GERMINUS, UPM,UMP,YACO,ANDAGO. *Estudio de herramientas de certificación de madurez de procesos de desarrollo.* California : s.n., 2006.
8. Araujo, Yuriana,Hilda López,Alexander Mendoza ,LuisTorrealba ,German Ortiz ,Roberto Guerra. Metodología RUP. *Metodología RUP.* [Online] mayo 2010. [Cited: diciembre 5, 2010.] http://www.scribd.com/doc/31440864/Metodologia-RUP...
9. Manuel Calero Solís. *Una explicación de la programación extrema (XP).* Madrid, España : s.n., 2003.
10. Carmina Lizeth Torres Flores, Germán Harvey Alférez Salinas. *Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa usando OpenUP.* Navojoa : s.n., 2008.
11. Enrique Hernández Orallo. *El Lenguaje Unificado de Modelado (UML).* 2002.
12. Larman, Craig. *UML y Patrones.* 2003.
13. International, Visual Paradigm. Visual Paradigm for UML. *Visual Paradigm for UML.* [Online] [Cited: diciembre 20, 2010.] http://www.visual-paradigm.com..
14. buenas tareas. buenas tareas. [Online] septiembre 10, 2010. [Cited: diciembre 20, 2010.] http://www.buenastareas.com/ensayos/Rational-Rose/717045.html..
15. ERICK ALEXANDER BENITEZ MORALES. *Herramienta Case "ArgoUML".* Michoacán de Ocampo, México : s.n., 2006.
16. Comunidad NetBeans. [Online] 2010. [Cited: diciembre 5, 2010.] http://netbeans.org/community/releases/68/..
17. Javier Garcia, José Ignacio Rodríguez,Iñigo Migno,Aitor Imaz,Alfonso Brazáles,Alberto Larzabal, Jesús Calleja,Jon García. *Aprenda JAVA como si estuviera en primero.* San Sebastián : s.n., 2000.

18. Sitio Oficial de Java. Sitio Oficial de Java. [Online] 2009. [Cited: diciembre 20, 2010.] http://java.com/es...
19. PHP. Hypertext Preprocessor. The PHP Group. *The PHP Group.* [Online] 2001-2011. [Cited: enero 8, 2011.] http://www.php.net/..
20. Ciberaula. *Ciberaula.* [Online] [Cited: diciembre 20, 2010.] http://java.ciberaula.com/articulo/tecnologia_java/. B-84140375..
21. Java.sum.com. *Java.sum.com.* [Online] [Cited: enero 22, 2011.] http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html ..
22. Capitulo 6. *Capitulo 6.* [Online] [Cited: febrero 8, 2011.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo6.pdf..
23. Spring Source. *Spring Source.* [Online] [Cited: febrero 8, 2011.] http://static.springsource.org/spring-security/site/..
24. Rosés, Albiol Francesc. *Introducción a Hibernate.* 2006.
25. iText Software Corp. (California, USA) and 1T3XT BVBA (Ghent, Belgium). PROGRAMAS-GRATIS.NET. *PROGRAMAS-GRATIS.NET.* [Online] [Cited: mayo 23, 2011.] http://itext.programas-gratis.net/..
26. Sencha. *Sencha.* [Online] [Cited: febrero 2, 2011.] http://www.sencha.com/products/extjs/license/..
27. Mateu, Carlos. *Desarrollo de aplicaciones web.* Barcelona : s.n., 2004. B-7.599-2004.
28. Pressman, Roger. *Ingeniería de Software, Un Enfoque Práctico.* s.l. : 5ta Edición, 2002. 8448132149.
29. Oktaba, Hanna. Introducción a Patrones. *Introducción a Patrones.* [Online] Facultad de Ciencias, UNAM. [Cited: abril 7, 2011.] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html..
30. Eneisy Osorio, Asdrubal Torres. *Propuesta de Arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito.* Ciudad de La Habana : Capítulo 2 y 3 , 2010.
31. Gutierrez, Jorge A. Saavedra. El Mundo Informático. *El Mundo Informático.* [Online] mayo 8, 2007. [Cited: mayo 24, 2011.] http://jorgesaavedra.wordpress.com/category/patrones-grasp/..
32. Pavarotty. Ingeneria de Software. *Ingeneria de Software.* [Online] noviembre 27, 2008. [Cited: mayo 24, 2011.] http://clases3gingsof.wetpaint.com/page/Artefactos+Arquitectura+de+Software..
33. Dania Mamani Nina, Daniela García Quispe,Mayra Melania Sotes Carrillo. *Artefacto:Diagrama de componentes.* La Paz, Bolivia : s.n., 2009.
34. Marcos Guglielmetti, Analia Lanzillotta, Guillem Alsina, David Yanover. MasterMagazine. *MasterMagazine.* [Online] febrero 11, 2005. [Cited: mayo 24, 2011.] http://www.mastermagazine.info/termino/4328.php..
35. Herranz, Raúl. Utópica Informática. *Utópica Informática.* [Online] diciembre 23, 2010. [Cited: mayo 24, 2011.] http://utopicainformatica.blogspot.com/2010/12/convencionesestandares-de-codificacion.html..
36. Linet Lores Sánchez, Diana Monné Roque. *Aplicación de las pruebas de liberación al sistema informático Genética Médica.* La Habana : s.n., 2009.

BIBLIOGRAFÍA

1. Definición.org. Lenguaje de Programación. [En línea] 20 de enero de 2011. http://www.definicion.org/lenguaje-de-programacion.

2. Digital, Contenido. Hibernate una Herramienta Mapping. [En línea] [Citado el: 8 de febrero de 2011.] http://contenidodigital.wordpress.com/2008/09/19/hibernate-una-herramienta-de-mapping-ii/.

3. Larman, Craig. UML y Patrones. 2003.

4. Landazuri, BarbaraA Mcdonald. Definición de Perfiles en Herramientas de Gestión de Requisitos. Madrid: s.n., 2005.

5. Orallo, Enrique Fernández. El Lenguaje Unificado de Modelado (UML). 2002.

6. Rodríguez de la Fuente, Pérez, Carretero.Programación de Aplicaciones Web. 2003 : Thomson.

7. Rosés, Francesc Albiol. Introducción a Hibernate. 2003.

8. Roger S. Pressman, MacGraw-Hill.Ingeniería del Software.Un enfoque práctico. 2001.

9. Sommerville, Ian.Ingenieria del Software. 2005. ISBN: 8478290745.

10. Victor Anaya, Patricio Letelier. SmarTTrace: Una Herramienta para Trazabilidad de Requisitos en Proyectos basados en UML. Valencia. 2002.