Universidad de las Ciencias Informáticas

Facultad 3

Título: Herramienta para la determinación de la complejidad de los requisitos funcionales de software.

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

Autor: Yoandi Díaz Ramos

Tutor(es): Ing. Daimara Mustelier Sanchidrian
Ing. Tamara Rodríguez Sánchez

La Habana, 11 junio de 2013
"Año 55 de la Revolución"

Declaración de Autoría

Declaro por este medio que yo, Yoandi Díaz Ramos con carnet de identidad 89020415783, ser autor del presente trabajo de diploma y autorizo a la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio, así como los derechos patrimoniales, con carácter exclusivo.

Para que así conste firmo la presente declaración jurada de auditoría en La Habana a los ____ días del mes de junio del año 2013.

Autor:

_____

Yoandi Díaz Ramos

Tutores:

_____ _____

Ing. Daimara Mustelier Sanchidrian Ing. Tamara Rodríguez Sánchez

Datos de Contacto

Tutora

- **Nombre y apellidos:** Ing. Daimara Mustelier Sanchidrian
- **Correo electrónico:** dmustelier@uci.cu
- **Situación laboral:** Profesor
- **Institución:** Universidad de las Ciencias Informáticas (UCI)
- **Dirección:** Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros
- **Rol:** Analista principal del proyecto ERP-Cuba del Centro de Informatización de Gestión de Entidades (CEIGE)

Tutora

- **Nombre y apellidos:** Ing. Tamara Rodríguez Sánchez.
- **Correo electrónico:** trodriquez@uci.cu
- **Situación laboral:** Profesor
- **Institución:** Universidad de las Ciencias Informáticas (UCI)
- **Dirección:** Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros
- **Rol:** Analista principal del proyecto ERP-Cuba del Centro de Informatización de Gestión de Entidades (CEIGE)

"La magnitud de lo que logramos no depende de lo que tenemos sino de lo que seamos capaces de hacer."

Dedicatoria

A mis abuelos maternos Yolanda y Alberto por su apoyo incondicional durante todos estos años de estudio, por su amor y comprensión.

A mi mamá Aydamí, por ser madre, padre, familia y todo; por educarme, enseñarme y guiarme en el camino que con los años marcarían quien realmente soy.

A mi hermana Yoania, a quien le deseo que pueda realizar todos sus sueños.

A mi novia Leidy por darme su amor y su apoyo incondicional en los momentos buenos y malos.

A José Manuel y Alexander que a pesar de no ser mis padres siempre me han apoyado.

A todos, dedico este trabajo.

Agradecimientos

A toda mi familia por su constante apoyo y preocupación, en especial a mis abuelos y a mi mamá por estar siempre a mi lado, por permitirme tomar mis propias decisiones, por guiarme y apoyarme siempre.

A mi novia Leidy una de las personas más importantes en mi vida, por estar a mi lado y por hacerme feliz cada día.

A Yaima Oval, Emilio, Yaima Betty, Made, Olguita y al Muchacho del Solapín por todo su apoyo y amistad.

A mis amigos, que de una forma u otra han estado junto a mí y han compartido conmigo estos años de estudio.

A mi grupo, por todos los momentos maravillosos que pasamos juntos durante la carrera.

A mis tutoras, por su guía durante la realización de este trabajo y por confiar en mí durante todo este tiempo.

A todos, Muchas Gracias…

Índice de Contenidos

Resumen ..... - 1 -
Introducción 2
Capítulo 1: Fundamentación Teórica 6
1.1. ¿Qué es la Ingeniería de Requisitos? 6
1.2. Clasificación de los requisitos 7
1.3. Técnicas para la captura y validación de los requisitos 8
1.4. Panorámica de herramientas de Gestión de Requisitos 9
1.5. Modelo de desarrollo, lenguaje de modelado y herramienta CASE 12
1.6. Lenguaje de programación 14
1.7. Frameworks, tecnologías y herramientas de desarrollo 15
1.8. Patrón de arquitectura 18
1.9. Patrones de diseño 19
1.10. Pruebas de software 20
Capítulo 2: Análisis y Diseño 22
2.1 Método UCI para calcular la complejidad de los requisitos de software 22
2.2 Propuesta de solución 28
2.3 Modelo conceptual 28
2.4 Requisitos 30
2.5 Mecanismos de diseño 35
2.6 Diagramas de clases del diseño 35
2.7 Diagramas de secuencia 37
2.8 Diagramas de clases persistentes 38
2.9 Modelo de datos 39
2.10 Patrones de diseños utilizados 40
2.11 Validación del diseño propuesto 41

Capítulo 3: Implementación y Prueba 50
3.1 Modelo de implementación 50
3.2 Código fuente 53
3.3 Pruebas de software 55
Conclusiones Generales 65
Recomendaciones 65
Referencias Bibliográficas 66
Bibliografía 69
Glosario de Términos 70
Anexos 71

Resumen

La Gestión de Requisitos es un componente vital en el desarrollo de un proyecto de software dado que apoya las actividades de planeación del proyecto. El uso de herramientas para auxiliar la Gestión de Requisitos se ha convertido en un aspecto importante de la ingeniería, considerando el tamaño y la complejidad de cada proyecto, estas vienen siendo un eslabón esencial para lograr un producto con la calidad requerida y en el tiempo establecido.

En la actualidad, diversas son las compañías y empresas de desarrollo de software que tienen una Gestión de Requisitos ineficiente debido a que utilizan algún enfoque informal ,no determinan el valor de la complejidad de los requisitos mediantes parámetros medibles e ignoran los posibles riesgos que trae como consecuencia en el avance del proyecto. El objetivo de este trabajo es proponer el uso de una herramienta libre, personalizada para la Gestión de Requisitos y que determine la complejidad de los requisitos funcionales del software. La misma formará parte del Sistema de Administración de Requisitos (SARE) teniendo en cuenta las necesidades de la UCI.

La propuesta realizada fue validada en el Centro de Calidad para Soluciones Informáticas (Calisoft), demostrando que con dicha herramienta se posibilita aumentar el desempeño de los especialistas en la determinación de la complejidad de los requisitos funcionales de software. Además permite aumentar la estimación de tiempo y recursos, así como apoyar las restantes estimaciones que se realizan en la universidad.

Palabras claves: Complejidad de los Requisitos, Estimación, Gestión de Requisitos, Herramienta, Ingeniería de Requisitos.

Introducción

El desarrollo del software ha tenido un auge notable y se considera el factor fundamental de muchos sistemas productivos. Las compañías y empresas mundiales, dependen cada vez más de un software que les permita obtener mejores resultados, optimizando el tiempo y con menor costo. (1)

Los avances producidos en la última década aún son insuficientes y muchas de las necesidades y expectativas de los clientes y usuarios no son satisfechas. Uno de los principales problemas por lo que la mayoría de los proyectos no cumplen los tiempos establecidos y con los objetivos pactados es porque no se realiza de manera correcta la Ingeniería de Requisitos (IR) lo que implica errores comunes muy costosos de reparar que consumen gran cantidad de tiempo, permitiendo una inadecuada Gestión de Requisitos (GR).

La GR es fundamental para el éxito de los proyectos que se realizan en el sector de la informática, saber lo que se va a hacer, cómo se va a hacer y dejar constancia de ello, trae como resultado un proyecto con mayor calidad, menor costo y que cumpla con el plazo establecido, garantizando que no se produzcan problemas ni desviaciones en su realización.

La GR constituye una de las actividades de la IR, que no por ser la última deja de ser la más importante, una vez concluidas las actividades de elicitación, análisis, especificación y validación, estos están listos para calcular su complejidad, partiendo de las especificaciones correctas de los requisitos que describen de que forma incide cada variable sobre la complejidad de los requisitos funcionales.

Muchos son los sistemas que gestionan requisitos en el mundo y sin embargo carecen de una herramienta que permita determinar la complejidad de los requisitos funcionales del software, paso tan importante como este para lograr una exitosa IR.

Como parte de la informatización de la sociedad cubana el país se encuentra inmerso en la producción de software para elevar el desarrollo de las empresas y contribuir con su perfeccionamiento empresarial. En ocasiones la premura, la mala planificación y los cortos plazos para elaborar un software han proporcionado que durante su desarrollo se incumpla o no se ejecute de manera adecuada la GR aunque esta sea necesaria y esté pactada en los cronogramas de desarrollo.

La Universidad de las Ciencias Informáticas (UCI), como entidad desarrolladora de software en su proceso de producción lleva a cabo las actividades de la IR para el desarrollo de sus proyectos. En el año 2010 surge en la UCI un método para la determinación de la complejidad de los requisitos, que posteriormente, en el 2012 emerge su nueva versión porque los datos ofrecidos eran insuficientes y presentaba algunas deficiencias quedando pendiente en esta última la herramienta a emplear para el cálculo de la complejidad.

La nueva versión abarca las variables que se consideran que aportan complejidad, solo que aún el cálculo de la complejidad es mediante una hoja de cálculo que presenta deficiencias como:

- Pertenece al paquete de Office donde solo funciona sobre plataformas privativas que atentan contra la soberanía tecnológica a la cual aspira el país.
- El trabajo se puede tornar engorroso dado que no todas las celdas presentan la formulación, llevando al usuario a realizar los cálculos fuera de la herramienta.
- Si se deja una celda en blanco realiza una división por cero que no tiene definición, originando estimaciones erróneas.

Debido a la problemática antes expuesta emerge el siguiente ***problema a resolver****:*

La determinación actual de la complejidad de los requisitos funcionales de software a través de la herramienta existente, influye negativamente en el desempeño de los expertos al realizar esta tarea en la Universidad de la Ciencias Informáticas.

El ***objeto de estudio*** lo constituyen las herramientas para la gestión de los requisitos definiendo como ***campo de acción*** las herramientas para determinar la complejidad de los requisitos funcionales de software.

Teniendo en cuenta el problema anterior, el ***objetivo general*** de este trabajo es desarrollar una herramienta que permita la determinación de la complejidad de los requisitos funcionales de software basado en la nueva versión del método UCI.

Con el fin de darle cumplimiento al objetivo general se desglosaron los siguientes ***objetivos específicos***:

- Realizar un estudio del estado del arte para la elaboración del marco teórico alrededor del objeto de estudio.

- Desarrollar la Ingeniería de Requisitos para la identificación de las funcionalidades y características de la herramienta.
- Diseñar la estructura de componentes, clases, datos aplicando los patrones de diseño, algoritmos y técnicas consideradas como necesarias en el estudio.
- Implementar la herramienta siguiendo las técnicas de programación estudiadas en la plataforma de desarrollo seleccionada.
- Validar la herramienta mediante la ejecución de técnicas y métricas aplicables al desarrollo de la solución.

Teniendo en cuenta la situación problémica antes expresada, se tiene como ***idea a defender*** que si se desarrolla una herramienta para determinar la complejidad de los requisitos funcionales de software entonces, se mejorará el desempeño de los expertos al calcular la complejidad de los requisitos funcionales de software en los proyectos de la universidad.

Los ***métodos científicos*** aplicados para llevar a cabo la investigación fueron los siguientes:

Métodos teóricos

Histórico-Lógico: para comprobar teóricamente como se ha comportado el proceso de la Ingeniería de Requisitos a lo largo de su desarrollo y 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.

Analítico-Sintético: para realizar el análisis de documentos permitiendo la obtención de elementos importantes que se relacionan con la complejidad de los requisitos.

Inductivo-Deductivo: para realizar generalización de la teoría estudiada y llegar a conclusiones específicas.

*Modelación***:** es empleado durante la fase de análisis y de diseño para elaborar los artefactos establecidos en el modelo de desarrollo.

Métodos empíricos

Entrevistas: a especialistas y líderes de proyecto con conocimientos sobre la GR.

Como ***posible resultado*** se tiene la realización de una herramienta para la determinación de la complejidad de los requisitos funcionales de software que formará parte del Sistema de Administración de Requisitos (SARE) desarrollado en Calisoft.

Estructura del documento

El presente trabajo de diploma se encuentra estructurado de la siguiente forma: introducción, tres capítulos, conclusiones generales, recomendaciones, referencias bibliográficas, bibliografía, anexos y glosario de términos.

Capítulo 1 Fundamentación Teórica: en este capítulo se realiza la fundamentación teórica mostrando aspectos fundamentales relacionados con la Ingeniería de Requisitos y la clasificación de estos. Se argumentan las técnicas para la captura y validación de los requisitos y se realiza un estudio de los sistemas de gestión de requisitos en el mundo, en Cuba y en la UCI. Muestra además el modelo de desarrollo que servirá de guía para la elaboración del trabajo y todas las herramientas, lenguajes y patrones que sustentarán el desarrollo de la herramienta. Por último explica las pruebas que serán aplicadas una vez terminado el trabajo.

Capítulo 2 Requisitos, Análisis y Diseño: en este capítulo se realiza el modelo conceptual propuesto y se definen los requisitos a implementar partiendo del estudio del método para determinar la complejidad de los requisitos. Se explican los patrones de desarrollo de software utilizados y el modelo de datos. Además se presentan los diagramas de clases del diseño, los diagramas de interacción y las métricas para validar el diseño.

Capítulo 3 de Implementación y Prueba: se muestra el diagrama de componentes que se elaboró durante la implementación de la herramienta, se describen implementaciones relevantes y se muestra el diagrama de despliegue, con el objetivo de tener ideas claras de cómo quedará la herramienta una vez desplegada. Además se detallan pruebas realizadas al software para comprobar su correcto funcionamiento.

Capítulo 1: Fundamentación Teórica

Introducción

En este capítulo se abordan los fundamentos teóricos asociados a la solución, se realiza un estudio del estado actual de la IR así como sus principales conceptos y características. Se aborda todo lo relacionado con la GR, sus tareas principales y descripción de las actividades que se desarrollan en ella. Por último se ofrece una panorámica general relacionada con herramientas de gestión de requisitos en Cuba y a nivel mundial; así como la descripción de las tecnologías utilizadas para el desarrollo de la solución propuesta.

1.1. ¿Qué es la Ingeniería de Requisitos?

La IR cumple un papel primordial en el proceso de producción de software, tiene como 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.

Varios fueron los conceptos relacionados con el tema de IR, propuesto por diferentes autores, según sus puntos de vista. Según Amador Duran Toro en su tesis 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)

Esta tiene como principal objetivo proporcionar un modelo de las necesidades del problema que debe resolverse de forma clara, coherente, precisa y no ambigua. (3)

El ciclo de vida de la Ingeniería de Requisitos comprende las siguientes actividades: (3)

- Elicitación: identificación de las fuentes de información relacionadas con el sistema y descubrir los requisitos de este.
- Análisis: estudio de los requisitos obtenidos, sus superposiciones y conflictos.
- Especificación: en esta actividad se documentan los requisitos acordados con el cliente, en un nivel apropiado de detalle.
- Validación: comprobación de que los requisitos obtenidos corresponden a lo que realmente se necesita por parte de los interesados.
- Gestión: control de los cambios en los requisitos.

Para el autor la investigación estará enmarcada en el siguiente concepto: los requisitos 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. Clasificación de los requisitos

Los requisitos de software generalmente pueden clasificarse en dos grandes grupos: los funcionales y los no funcionales. Cuando se habla de un requisito funcional se hace referencia a aquellos requisitos que describen la funcionalidad o los servicios que se espera que el sistema provea, sus entradas y salidas, excepciones, estos dependen del tipo de software y del sistema que se desarrolle y de los posibles usuarios del software. Los requisitos no funcionales se refieren a las propiedades emergentes del sistema como la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, la capacidad de los dispositivos de entrada/salida, y la representación de datos que se utiliza en las interfaces del sistema (3).

Existen otras clasificaciones de requisitos que tienen en cuenta principalmente el impacto de estos sobre el desarrollo del producto. Dentro de las mismas se pueden encontrar la clasificación de prioridad que se refiere a la prioridad, como la característica que describe la importancia que tiene un requisito en términos de implementación. Esta clasificación permite definir la secuencia en que ocurrirán las actividades de diseño y prueba de cada requisito. Otra clasificación que reciben los requisitos es la complejidad de un requisito que describe la dificultad de diseño e implementación del mismo dentro del proceso de desarrollo de software. (4)

Según la complejidad de un requisito, esta se puede clasificar en baja, media y alta. Es importante para los expertos clasificar los requisitos de software según su complejidad, porque

permite brindar un mejor tratamiento de los mismos durante el desarrollo de cualquier producto, de manera que la fuerza de trabajo estaría enfocada según la clasificación por complejidad establecida. Solamente de esta forma se puede tener dominio de hasta dónde se puede llegar y los recursos tanto humanos, como materiales que serían necesarios para el desarrollo del producto. (4)

Específicamente esta clasificación será la base para el desarrollo de la herramienta propuesta en esta investigación.

1.3. Técnicas para la captura y validación de los requisitos

La identificación de los requisitos que debe cumplir un software es una actividad que se desarrolla al iniciarse cada sistema. En este proceso los analistas extraen de diferentes fuentes de información los datos que son necesarios para conocer las funcionalidades que implementará el sistema, haciéndose necesario por parte de los analistas el empleo de técnicas que permitan establecer una buena comunicación con los interesados del producto y así lograr la satisfacción del cliente. Las técnicas a utilizar para la captura de requisitos son:

Entrevistas: es una de las técnicas más usadas en la captura de requisitos. Consiste en establecer una conversación entre personas de ambas partes. Estas son aplicadas a los especialistas funcionales.

Tormenta de ideas: reunión de varios interesados en la que todos expresan sus ideas sobre el problema y su solución. La forma de llevarla a cabo es que cada participante diga su idea sin ser interrumpido por otro. Al finalizar la sesión de lluvia de ideas se puede hacer una recolección de ideas sin duplicidad.

Sistemas existentes: fue utilizada durante el análisis de varios sistemas existentes que estén relacionados con el que va hacer construido. Analizándolos en cuanto a un grupo de características necesarias que debe llevar el que se va a elaborar.

También existen técnicas para la validación de los requisitos las cuales tienen como objetivo general demostrar que su definición es la que el usuario final necesita. (5)

Muchas son las técnicas para validar requisitos, pero las utilizadas durante el desarrollo del trabajo fueron:

Revisión Técnica Formal: son las revisiones realizadas por el usuario final del sistema y especialistas con el objetivo de validar la especificación de requisitos permitiendo detectar deficiencias, ambigüedades, omisiones y errores, tanto de formato como de contenido. (6)
Auditorías: consisten en un chequeo de los resultados contra una lista de chequeo predefinida o definida a comienzos del proceso.
Prototipos de Interfaz de Usuario: ayudan a identificar, comunicar y probar un producto antes de crearlo. La realización de los mismos, logra un entendimiento común entre el cliente y los desarrolladores, solucionando así la mayoría de los cambios posteriores en los proyectos de desarrollo de software. (31)

1.4. Panorámica de herramientas de Gestión de Requisitos

En la actualidad, las compañías y empresas intentan producir software con buena calidad para satisfacer las necesidades de los clientes y que este obtenga el producto esperado, para lograrlo es necesario la utilización de sistemas informáticos para la GR. (7)

Herramientas informáticas que gestionan requisitos

Las organizaciones con mejores posiciones en el mercado tienden a incrementar el uso de este tipo de herramienta, las más usadas a nivel mundial se mueven entre propietarias, libres y multiplataformas, destacándose las siguientes:

- Requisite Pro
- REM
- Caliber RM
- DOORS
- IRqA
- OSRMT
- GatherSpace

A continuación se exponen algunas características de estas herramientas que gestionan requisitos: (1) (7)

OSRMT v1_5: es una herramienta de software libre, el nombre completo es Open Source Requeriment Management Tool. Fue diseñada para servir el ciclo de vida completo del desarrollo del software. Es independiente de la plataforma que se vaya a utilizar, pues corre sobre Java.

Permite la descripción de los requisitos, además de los documentos relacionados con la Ingeniería de Requisitos, también comprende las distintas matrices de trazabilidad, funcionalidades, casos de usos y casos de pruebas.

GatherSpace: es una herramienta de gran alcance, con toda la gerencia y el uso en línea simple de los requisitos que permite centralizar, modelar y compartir requisitos del software. Es capaz de proporcionar una intuitiva solución para pequeñas y grandes organizaciones. Es capaz de gestionar los requisitos del software utilizando las especificaciones funcionales, además de los casos de usos así como el modelado. Es un software no-instalable, está en línea 100% de las funcionalidades, solo se necesita un navegador.

RequisitePro: es una herramienta centrada en documentos, se almacenan los requisitos asociándolos a documentos (aunque también permite guardarlos directamente en la base de datos).Auxilia especialmente el control de cambio de requisitos, con trazabilidad para especificaciones de software y pruebas. Está muy unido a MS Word ya que es parte de Microsoft Development. La herramienta permite el uso de Oracle sobre Unix o Windows como “back-end database” y también soporta SQL Server sobre Windows.

CaliberRM: es para sistemas grandes y complejos, proporciona una base de datos de requisitos con trazabilidad. La compañía ve a los requisitos como parte del proceso de gestión de la calidad del software al igual que las pruebas y el trazado de defectos. Caliber está basado en Internet y maneja referencia de documentos, responsabilidad de usuario, trazabilidad, prioridad y estado entre algunas otras características.

DOORS: a diferencia del resto de las herramientas, considera los requisitos como objetos y los documentos como módulos. Tiene una orientación basada en objetos, frente a RequisitePro y CaliberRM, que manejan solamente requisitos y sus atributos. Es una herramienta para organizaciones grandes que necesitan controlar gran cantidad de usuarios y requisitos de sistemas con una completa trazabilidad. Proporciona buena visualización de tales documentos como jerárquicos y su lenguaje de extensión, permite una gran variedad de soporte de herramientas a ser construidas.

IRqA 3.0: herramienta CASE de Ingeniería de Requisitos, diseñada para soportar las actividades realizadas en el proceso de especificación de sistemas. Proporciona y formaliza la comunicación

entre cliente y proveedor y entre los distintos miembros del equipo de desarrollo. Facilita la captura, organización y análisis de los requisitos y la especificación de la solución mediante un apoyo metodológico adaptable a cada cliente.

Realizando un estudio más profundo de cada una de estas herramientas se elaboró una tabla comparativa teniendo en cuenta ocho parámetros y su comportamiento en cada uno de ellos.

1. Tipos de licencias bajo la que se publica el producto y que sistemas operativos soporta.
2. Captura e identificación de requisitos.
3. Captura de la estructura de los elementos del sistema.
4. Análisis de trazabilidad.
5. Gestión de la configuración.
6. Ambiente del sistema.
7. Soporte y mantenimiento.
8. Complejidad de los Requisitos.

Tabla 1. Comparación entre las herramientas seleccionadas

Herramientas	1	2	3	4	5	6	7	8
OSMRT	x	x	x	x	x	x	x	-
GatherSpace	-	x	x	x	x	x	x	-
RequisitePro	-	x	x	x	x	x	x	-
CaliberRM	-	x	x	x	x	x	x	-
DOORS	-	x	x	x	x	x	x	-
IRqA	-	x	x	x	x	x	x	-

Sistemas informáticos cubanos que gestionan requisitos

En Cuba, de manera general las empresas productoras de software utilizan herramientas informáticas para la GR evidenciándose en los últimos años, pero aún, existen dificultades por problemas de compatibilización de las mismas con los sistemas operativos utilizados en su elaboración, las licencias o por desconocimiento en el uso de las herramientas.

Algunas empresas cubanas pese a que conocen la existencia de herramientas que facilitan la GR no han dado el salto hacia su utilización y otras ni siquiera han modificado o adaptado las existentes para su uso en la organización.

En entrevistas realizadas a líderes de proyectos y especialistas en la actividad de la GR en proyectos de la UCI y empresas asociadas a esta actividad pudimos corroborar la necesidad de la utilización, implementación o adaptación de algunas de estas herramientas informáticas porque ninguna incluye la funcionalidad del cálculo de la complejidad de los requisitos funcionales del software.

Por ello, Calisoft perteneciente al Ministerio de la Informática y las Comunicaciones (MIC) desarrolla un sistema para las empresas cubanas que lleva por nombre Sistema para Administración de Requisitos (SARE). Este sistema realiza la gestión de requisitos y la automatización o flexibilización del proceso de trazabilidad de los mismos para un mejor control, menor esfuerzo personal y una estandarización de las actividades de la IR.

Después de analizar las herramientas que gestionan requisitos y conocer las funcionalidades que cada una ofrece según los parámetros de comparación, se puedo apreciar que ninguna de estas realiza el cálculo de la complejidad de los requisitos funcionales del software. Por lo tanto se trabajará con el sistema SARE que cuenta con pasos de avances en esta temática y que aún se encuentra en perfeccionamiento permitiendo nuevos desarrollos e inclusiones de nuevas funcionalidades.

1.5. Modelo de desarrollo, lenguaje de modelado y herramienta CASE

1.5.1 Modelo de desarrollo

El modelo de desarrollo a utilizar es el propuesto por el Centro de Informatización de la Gestión de Entidades (CEIGE) en su versión 1.1. Detalla el ciclo de vida de sus proyectos con la incorporación de los distintos subprocesos dictados por CMMI (Capability Maturity Model Integration, por sus siglas en inglés) para su nivel II, certificación obtenida por el centro en julio de 2011 y reconocida por el Instituto de Ingeniería de Software (SEI, por sus siglas en inglés) como aval de la calidad de su proceso de desarrollo de software. (8)

A continuación se describen las disciplinas por las que se transitará en la elaboración de la herramienta y los artefactos que se generarán en cada una de ellas, teniendo en cuenta el flujo definido dicho modelo de desarrollo.

- Disciplina de Requisitos: para dar inicio al trabajo de esta disciplina se elaborará un modelo de dominio teniendo en cuenta la investigación que le antecedió a esta investigación. Además se realizarán las especificaciones de los requisitos.
- Disciplina de Análisis y Diseño: se construirán los diagramas de clases del diseño, el modelo de datos, los diagramas de secuencia y clases persistentes.
- Disciplina de Implementación: se confeccionará el diagrama de componentes, el diagrama de despliegue y el código fuente de la herramienta.

1.5.2 Lenguaje Unificado de Modelado

UML (Unified Modeling Language): es un lenguaje que permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Es un conjunto de herramientas, que permite modelar, analizar y diseñar sistemas orientados a objetos. (9)

Se ha convertido en el estándar de factor de la industria. UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, puesto que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otra rama. UML cuenta con varios tipos de diagramas que son la representación gráfica de un conjunto de elementos y sus relaciones visualizan el sistema desde diferentes perspectivas. (10).

1.5.3 Herramienta CASE

La ingeniería de software asistida por computadora (CASE: Computer Aided Software Engineering: Ingeniería de Software Asistida por Computación) ayuda a los ingenieros de software en todas las actividades asociadas a los procesos de software. Las herramientas CASE automatizan las actividades de gestión de proyectos, gestionan todos los productos de los trabajos elaborados a través del proceso y ayudan a los ingenieros en el trabajo de análisis, diseño y codificación. La importancia de su uso radica en permitir reducir la cantidad de esfuerzo que se requiere para producir un producto o para alcanzar un hito en el proceso de desarrollo. Además, contribuyen a la calidad del software dado que proporcionan nuevas formas de observar la información de la ingeniería del software. (11)

Visual Paradigm para UML: es una herramienta CASE multiplataforma de modelado visual UML, muy potente y fácil de utilizar. Soporta el ciclo de vida completo del desarrollo de software, ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. (12)

Dentro de las principales características que posee esta herramienta está el soporte de UML en su versión 2.0, la disponibilidad de integrarse en los principales IDEs. Además soporta una gama de lenguajes en la Generación de Código e Ingeniería Inversa en Java, C++, CORBA IDL, PHP, Ada y Python. Permite realizar diagramas de procesos de negocio, diagramas de flujo de datos entre otros. Realiza la distribución automática de diagramas e importación y exportación de ficheros así como la edición de figuras. (13)

1.6. 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 este pueda comunicarse con los dispositivos de hardware y software existentes.(14)

Se estudiaron varios lenguajes de programación entre los que se encuentran: PHP5, Python, Java, C++ y C#, sin embargo el lenguaje a utilizar es Java pues la herramienta a desarrollar formará parte del sistema SARE el cual está implementado en Java.

Java: es un lenguaje de programación sencillo, orientado a objetos, de propósito general e independiente de la plataforma de desarrollo. (15)

Las características principales respecto a otros lenguajes de programación son: (16)

Simple: ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.

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 es interpretado, un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java.

Arquitectura neutral: compila su código a un fichero objeto de formato independiente a la arquitectura de la máquina en que se ejecutará, de esa manera logra ser un lenguaje que no depende de una arquitectura computacional definida. (17)

Dinámico: no requiere que se compilen todas las clases de un programa para que este funcione.

Orientado a objetos: 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. (18)

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. (19)

1.7. Frameworks, tecnologías y herramientas de desarrollo

La selección correcta de las herramientas, tecnologías y frameworks que se utilizan en el desarrollo de un software se traduce en ahorro de tiempo y trabajo dentro de cualquier proyecto. A continuación se realiza una breve descripción de las que serán utilizadas en el desarrollo del trabajo, partiendo de la particularidad que la herramienta a desarrollar forma parte del sistema SARE y debe cumplir con las políticas establecidas.

1.7.1 Frameworks

En el desarrollo de la arquitectura base del sistema se utilizaron los framework de aplicación Spring y Spring-Security y como framework de soporte Hibernate. Estos son muy usados por sus múltiples ventajas en el desarrollo de aplicaciones web en el lenguaje Java. Además Ext JS, utilizado como framework de presentación para crear interfaces de manera fácil y sencilla.

Spring 3.0.2: es un framework de aplicación desarrollado 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. 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. (20)

Spring-Security 3.0.5: es un framework de autenticación potente, altamente personalizable y con un marco de control de acceso. Es fácil de aprender, implementar y administrar. Dentro de las ventajas principales posee la gestión de seguridad a varios niveles, separa la lógica de aplicaciones de la seguridad del sistema y soporta muchos modelos de identificación de los usuarios. Soporta diversos modelos de identificación de los usuarios (HTTP BASIC, HTTP Digest, basada en formulario, LDAP, OpenID y JAAS). Además los mecanismos de seguridad pueden ser ampliados implementando clases propias que extiendan el modelo de Spring Security. (21)

Ext JS 3.3.1: es un framework 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. 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. (22)

Hibernate 4.1.3: es un framework que permite el mapeo objeto-relacional y más conocido por sus siglas en inglés Object-Relational Mapping (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 ventaja permite el soporte para múltiples dialectos, es decir, puede mapear diferentes sistemas gestores de bases de datos, entre los que se encuentra Postgres. (23)

1.7.2 Entorno de Desarrollo Integrado

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 usados a nivel mundial se encuentran el Eclipse y el Netbeans. (24)

Como la herramienta a elaborar formará parte del sistema SARE se utilizará como entorno integrado de desarrollo el Netbeans en su versión 6.9.

NetBeans 6.9: es una 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. Soporta varios lenguajes, entre ellos: Java, C++, Ruby on Rails y PHP. Posee una instalación y actualización simple con un diseñador visual de formularios para Swing GUI. Además tiene características visuales para el desarrollo web y un creador gráfico de juegos para celulares. Es fácilmente integrable con la herramienta CASE Visual Paradigm. (25)

1.7.3 Gestor de Base de Datos

PostgreSQL 9.1.0 es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS). Este está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo. El proyecto PostgreSQL sigue actualmente un activo proceso de desarrollo a nivel mundial gracias a un equipo de desarrolladores y contribuidores de código abierto. (27)

Este proporciona un gran número de características como aproximación de los datos a un modelo objeto-relacional y es capaz de manejar complejas rutinas y reglas. Es altamente extensible soportando operadores, funciones, métodos de acceso y tipos de datos definidos por el usuario. Soporta lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL y permite la gestión de diferentes usuarios, así como también los permisos asignados a cada uno de ellos. Además incluye herencia entre tablas. (28)

1.7.4 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.

Apache Tomcat es una aplicación de código abierto de Java Serverlet 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. Es un servidor gratis, robusto, estable, fácil de instalar y es compatible con las API más recientes de Java. Su extensibilidad y construcción modular se pueden poner módulos para ampliar su funcionalidad. Ocupa muy poco espacio, teniendo su código binario (todo clases de Java) en un tamaño total de apenas un megabyte, de modo que no es raro que se ejecute tan deprisa. (29)

1.7.5 Navegador

Mozilla Firefox es un navegador de código abierto, multiplataforma, basado en el código de base de Mozilla que proporciona una navegación más rápida, segura y eficiente que otros navegadores. Entre sus principales características se encuentran la velocidad, la seguridad y se pueden usar varios perfiles de usuario con configuraciones diferentes para cada uno.

Este navegador consta además de un innumerable conjunto de extensiones y temas dentro de su paquete de complementos, que le permiten modificar la versión original instalada y personalizarla más al usuario, así como brindarle mayores comodidades. En el desarrollo de aplicaciones web es muy utilizado el complemento firebug, a través del cual los desarrolladores pueden llevar un control del tráfico de información desde el cliente hasta el servidor, brindando grandes facilidades a la hora de conectar la programación de la capa de presentación con la de negocio. (30)

1.8. Patrón de arquitectura

El patrón de arquitectura que se utilizará para el desarrollo de la herramienta será el Modelo-Vista-Controlador (MVC) fundamentándose en el framework Spring 3.0.2 que lo implementa internamente. Este separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario; es decir separa en tres capas diferentes los datos de una aplicación, la interfaz de usuario, y la lógica de control:

Modelo: administra el comportamiento y los datos del dominio de la aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: maneja la visualización de la información, presentando el modelo en un formato adecuado para interactuar, que usualmente es la interfaz de usuario.

Controlador: controla el flujo de datos entre la vista y el modelo; respondiendo a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. (31).

1.9. Patrones de diseño

Los patrones de diseño surgen ante la necesidad de la reutilización de diseños y de soluciones a problemas frecuentes encontrados por los ingenieros. Con la reutilización se consigue la reducción de tiempos y la disminución del esfuerzo de mantenimiento, esto trae consigo mayor eficiencia y consistencia en el diseño de la solución.

Para lograr una correcta construcción del diseño de la herramienta para la complejidad de los requisitos se hace necesario realizar un estudio previo de los patrones de diseño a utilizar. Estos contribuirán a la descripción de clases y objetos que se comunican entre sí. Los patrones que serán utilizados en el diseño de la herramienta son los patrones GRASP. (32)

Patrones GRASP (General Responsibility Assignment Software Patterns**):** son patrones generales de software para asignación de responsabilidades. Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

Creador: se asigna la responsabilidad de que una clase B cree un objeto de la clase A solamente cuando:

- B contiene a A.
- B es una agregación (o composición) de A.
- B almacena a A.
- B tiene los datos de inicialización de A (datos que requiere su constructor).
- B usa a A.

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulamiento y reutilización.

Bajo Acoplamiento: promueve la baja dependencia y una alta reutilización de las clases. Asigna responsabilidades manteniendo un bajo acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de cambios que acrecientan la reutilización. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda.

Experto: la responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.

Controlador: asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase específica.

Alta Cohesión: la alta cohesión al igual que el bajo acoplamiento logran un diseño fácil de reutilizar y adaptar, más legibilidad para los programadores y diseñadores en general; la extensibilidad y flexibilidad del diseño aumentan, dado que las clases usan solo lo que necesitan y en su mayoría son recursos propios, por otro lado, cada cual hace lo que le corresponde hacer según la parte del problema que está modelando. (32)

1.10. Pruebas de software

Las pruebas de software son básicamente un conjunto de actividades dentro del desarrollo del software que involucran las operaciones del sistema bajo condiciones controladas y evaluando los resultados donde el único instrumento adecuado para determinar el status de la calidad de un producto software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. (33)

Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso en desarrollo asegurando la calidad del sistema con el objetivo principal de presentar la información sobre la calidad del producto a las personas responsables de estas. Todo este proceso de pruebas, sus objetivos y los métodos y técnicas usados se describen en el plan de prueba.

Pruebas de caja blanca: se realizan 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.

Conclusiones parciales

En este capítulo se abordaron elementos teóricos arribando a las siguientes conclusiones:

- La Ingeniería de Requisitos y la clasificación de los mismos.
- Con estudio de diferentes herramientas que gestionan requisitos se pudo concluir que estas presentan ventajas y desventajas de acuerdo a los parámetros medidos, siendo el sistema SARE el único con pasos de avances e indicadores medibles definidos para el cálculo de complejidad de los requisitos funcionales del software.
- Se implementará una herramienta que forma parte del sistema SARE, para el cálculo de complejidad de los requisitos utilizando como guía el modelo de desarrollo definido en el CEIGE.
- Las herramientas y tecnologías seleccionadas permitirán analizar, diseñar, implementar y validar la solución que se propone elaborar en los capítulos posteriores, puesto que son las más idóneas para trabajar dentro de la plataforma tecnológica de la Universidad teniendo en cuenta las características del sistema.

Capítulo 2: Análisis y Diseño

Introducción

En este capítulo se describe la propuesta de solución de la herramienta a desarrollar basada en la nueva versión del método UCI para el cálculo de la complejidad de los requisitos funcionales. Además se particulariza en las funcionalidades y características que tendrá la herramienta a desarrollar para el sistema SARE. Se presenta el modelo conceptual con las correspondientes descripciones de los conceptos asociados a la herramienta. Se traducen los requisitos a una especificación que describe cómo implementar la herramienta a través del diseño, enfocando el cumplimiento de los objetivos teniendo en cuenta los requisitos funcionales y no funcionales. Se realizan los diagramas de clases del diseño y los diagramas de secuencias más relevantes según el escenario de Requisitos y se muestra el modelo de datos. Por último se explica la arquitectura, los principales patrones y métricas de diseño utilizados.

2.1 Método UCI para calcular la complejidad de los requisitos de software

A continuación se exponen los elementos que contienen el método UCI que son la base para el desarrollo de la herramienta. (4)

Actividades que contiene el método y su forma de realización:

2.1.1 Definición de las variables

Las variables del método UCI para el cálculo de la complejidad de los requisitos son:

1. **Reutilización**: es la capacidad de definir el requisito para que sea compartido o utilizable en otras ocasiones y otros productos.
2. **Facilidad de cambio**: esfuerzo específico de diseño e implementación para facilitar cambios futuros durante el desarrollo del producto.
3. **Objetivos especiales de seguridad**: dificultad de implementación en un determinado requisito para asegurar el cumplimiento de características especiales de seguridad como por ejemplo: no utilizar comunicaciones encriptadas (SSL) para un sitio web.
4. **Consultas externas**: los requisitos pueden presentar diversidad en la cantidad y complejidad de la interacción con la fuente de datos. Las posibles fuentes de datos

identificadas son las siguientes, consideradas subcomponentes e igualmente evaluadas individualmente como componentes: base de datos, ficheros y otros.

5. **Escalabilidad**: capacidad del enfoque para ajustarse a diferentes tipos de proyecto, tanto por su tamaño como por su misma naturaleza.
6. **Interfaces**: el componente interfaz se aplica a requisitos que presenten algún tipo de complejidad en su interacción con los siguientes elementos, considerados subcomponentes y a los que se le aplica la misma forma de evaluación que a los componentes:
   - *Humanas (formularios, informes)*
   - *Equipo (Tomógrafos, Rayos X); Ej. API, drivers*
   - *Programación (Programas externos necesarios para apoyar el producto); Ej. Adicionar un nuevo usuario, usa el LDAP*
   - *Comunicación (Protocolos de comunicación que serán utilizados); Ej. HL7*
   - *Atributos (Se refiere a la complejidad que puede agregarle la cantidad de atributos contenidos en una interfaz.)*
7. **Diferentes comportamientos**: un mismo requisito se comporta de manera diferente ante determinadas situaciones; Ej. Admisión de un paciente, puede ser normal o por emergencia, en el primer caso se recoge más información que en el segundo.
8. **Lógica de negocio**: los requisitos pueden presentar diferentes niveles de complejidad para la implementación de la lógica de negocio teniendo en cuenta los siguientes elementos:
   - *Flujo de datos lógico* (Hace referencia a la complejidad que puede encerrar la lógica de negocio en cuanto al número de eventos o pasos que se deben llevar a cabo para cumplir con un determinado requisito, incluyendo así la cantidad de flujos alternativos del requisito.)
   - *Patrones* (Complejidad en cuanto a la cantidad y diversidad de patrones a utilizar según la lógica de negocio para cumplir con un requisito.)
   - *Operaciones matemáticas* (Se refiere a la inclusión de métodos matemáticos o cálculos complejos necesarios para cumplir con un determinado requisito; Ej. Cálculo de facturas mensuales para un contrato.)
9. **Restricciones de validaciones**: complejidad y cantidad de validaciones que lleve un requisito, tanto las validaciones en el lado del cliente, como en el servidor.

10. **Dependencia con otros requisitos**: describe el grado de relación de dicho requisito con otros, según la cantidad de requisitos de los cuales depende. (4)

2.1.2 Aplicación del método UCI (4)

1. Asignación de pesos a cada variable: a todas las variables que se evaluarán en la nueva versión del método se le asignó un peso del 1 al 5. Aunque algunas de ellas ya poseían un peso asignado en el método UCI, se hizo necesario volver a asignarle un peso nuevo, puesto que esta vez dicha asignación se consolidó a partir de analizar la cuarta pregunta de la encuesta que realiza la autora Tatiana García Mirabal y no a través de un panel de especialistas como en el método anterior, el resultado se muestra a continuación, ver Tabla 2. (4)

Tabla 2. Pesos de las variables

Variables	Peso
Reutilización	5
Facilidad de cambio	3
Objetivos especiales de seguridad	5
Consultas externas	5
Escalabilidad	5
Interfaces	3
Diferentes comportamientos	3
Lógica de negocio	5
Restricciones de validaciones	5
Dependencia con otros requisitos	4

2. Normalización del peso de cada variable: para obtener un mejor resultado al trabajar con el peso de cada variable se hace necesaria la normalización de estos, lo cual se realizará a través del método de normalización por suma, mediante la siguiente fórmula: (4)

$$w_n = \frac{C_n}{\sum_{i=1}^{k} C_i}$$

C_n Peso asignado a la variable

w_n Peso normalizado de cada variable

k Total de variables

Ejemplo:

Teniendo en cuenta la fórmula antes planteada para normalizar el peso de cada variable, se mostrará a continuación cómo se normalizó el peso de la variable Reutilización, como ejemplo ilustrativo: (4)

Cn (Reutilización)=5

$$\sum_{i=1}^{10} Ci = 5+3+5+5+5+3+3+5+5+4 = 43$$

Wn = 5/43 = 0.116279

Después de haber aplicado la normalización del peso de cada una de las variables, se obtuvo el siguiente resultado, ver Tabla 3.

Tabla 3. Peso normalizado de las variables

Variables	Peso	Peso normalizado
Reutilización	5	0.116279
Facilidad de cambio	3	0.069767
Objetivos especiales de seguridad	5	0.116279
Consultas externas	5	0.116279
Escalabilidad	5	0.116279
Interfaces	3	0.069767
Diferentes comportamientos	3	0.069767
Lógica de negocio	5	0.116279
Restricciones de validaciones	5	0.116279
Dependencia con otros requisitos	4	0.093023
Total	**43**	**1**

3. Definición de los parámetros para realizar la valoración de las variables por requisitos

Los parámetros que guiarán a los analistas en sus proyectos a la hora de valorar cada variable, se muestran a continuación: (4)

Tabla 4. Clasificación de las variables

Variables
Reutilización
Alta: cuando la descripción del requisito es muy genérica; Ej. Análisis de dominio.
Media: cuando esté en el término medio de que no sea ni muy genérico, ni muy específico.
Baja: cuando está destinado solo para ser utilizado en una solución determinada, es decir que es específico.
Facilidad de cambio
Alta: presenta una alta relación con otros requisitos o una gran complejidad en cuanto a la lógica del negocio.
Media: presenta una mediana complejidad en cuanto a la relación con otros requisitos y la lógica del negocio.
Baja: cuando no presenta relación con otros requisitos y su lógica de negocio es simple.
Objetivos especiales de seguridad
Alta: si es un requisito funcional completamente de seguridad.
Media: tiene que tener en cuenta algunos aspectos relacionados con la seguridad que deben cumplirse.
Baja: no está relacionado con la seguridad que debe brindar el proyecto de software.
Consultas externas
Alta: contiene cuatro o más consultas a las distintas fuentes de almacenamiento.
Media: contiene más de una y menos de cuatro consultas a las distintas fuentes de almacenamiento.
Baja: contiene una o ninguna consulta a las distintas fuentes de almacenamiento.
Escalabilidad
Alta: cuando el requisito representa a una funcionalidad muy específica del sistema.
Media: cuando el requisito representa a una funcionalidad en término medio en cuanto a su particularidad para un sistema.
Baja: cuando el requisito es una funcionalidad común de cualquier software. Ej. Adicionar, Modificar o Eliminar.
Interfaces
Alta: requiere la construcción de más de cuatro interfaces. Teniendo en cuenta que uno de los parámetros en que se descompuso esta variable es la cantidad de atributos contenidos, además debe cumplir para tener una complejidad alta que la cantidad de atributos en las interfaces en general supere o iguale a veinte atributos.
Media: requiere la construcción de tres o cuatro interfaces. Teniendo en cuenta que uno de los

parámetros en que se descompuso esta variable es la cantidad de atributos contenidos, además debe cumplir para tener una complejidad alta que la cantidad de atributos en las interfaces en general supere o iguale a quince atributos.
Baja: requiere la construcción de hasta dos interfaces. Teniendo en cuenta que uno de los parámetros en que se descompuso esta variable es la cantidad de atributos contenidos, además debe cumplir para tener una complejidad alta que la cantidad de atributos en las interfaces en general supere o iguale a cinco atributos.
Diferentes comportamientos
Alta: presenta cuatro o más formas de manifestarse.
Media: presenta dos o tres formas de manifestarse.
Baja: solo se puede realizar de una sola forma.
Lógica de negocio
Alta: cuando su lógica comprende más de diez pasos o incluye más de una operación matemática, o usa más de tres patrones.
Media: cuando incluye menos de diez pasos y además incluye un cálculo matemático de mediana complejidad, o la utilización de algún patrón.
Baja: no contiene operaciones matemáticas, ni utiliza algún patrón y la cantidad de pasos es menor que diez.
Restricciones de validaciones
Alta: presenta tres o más tipos de validaciones.
Media: presenta dos tipos de validaciones.
Baja: presenta un solo tipo de validación.
Dependencia con otros requisitos
Alta: depende de cuatro o más requisitos para su realización.
Media: depende de dos o tres requisitos como máximo para su realización.
Baja: depende solamente de un requisito o no presenta dependencia alguna.

4. Cálculo del valor de la complejidad final por requisito

Para calcular la complejidad se utilizará el método matemático de la suma ponderada que se rige por la siguiente fórmula: (4)

$$V_i = \sum_{i=1}^{k} w_i v_i$$

V_i Valoración final de la complejidad obtenida por el requisito n

v_i Valoración de cada variable del requisito i

w_n Peso normalizado de cada variable

A partir de la obtención de V_i se determina la evaluación en alta, media o baja de la complejidad del requisito n, según el rango propuesto por el método UCI, el cual se muestra a continuación: (4)

- Baja si, $0 < V_n \leq 1.4$
- Media si, $1.4 < V_n \leq 2.4$
- Alta si, $2.4 < V_n \leq 3$

2.2 Propuesta de solución

Durante el desarrollo de la investigación se realizaron entrevistas a especialistas y líderes de proyecto que coincidían que la GR era un ente fundamental en el desarrollo de un software y que hacer uso inadecuado de ella proporciona dificultades al buen desempeño del mismo. Además el 72% de los entrevistados utilizan una herramienta para gestionar requisitos no siendo así para el 28% restante. Sin embargo todos los especialistas y líderes que utilizan herramientas para la GR aluden que ninguna tiene incorporado el cálculo de la complejidad de los requisitos. (Ver Anexo 1)

Los entrevistados demuestran la utilización de SARE, muchos centran sus esfuerzos solo en algunos subsistemas de él, dado que los restantes se encuentran incompletos o en construcción.

SARE permitirá determinar la complejidad de los requisitos funcionales del software de mejor forma, sobre un entorno web, permitiendo la conectividad desde diferentes estaciones de trabajo. Se desarrollará sobre plataforma libre, cumpliendo las políticas de soberanía tecnológica a las que aspira el país. El usuario no tendrá acceso a dejar campos en blanco, ni a modificar ninguna fórmula generándose de forma automática con los datos introducidos por él.

Siguiendo las políticas de seguridad informática de la universidad, la herramienta contará con una previa autenticación de usuarios y los permisos para cada rol permitiendo interactuar con las diferentes funcionalidades y privilegios.

2.3 Modelo conceptual

El modelo conceptual es una representación visual de los conceptos del mundo real significativos para un problema. El modelo conceptual del presente trabajo se muestra en la Figura 1.

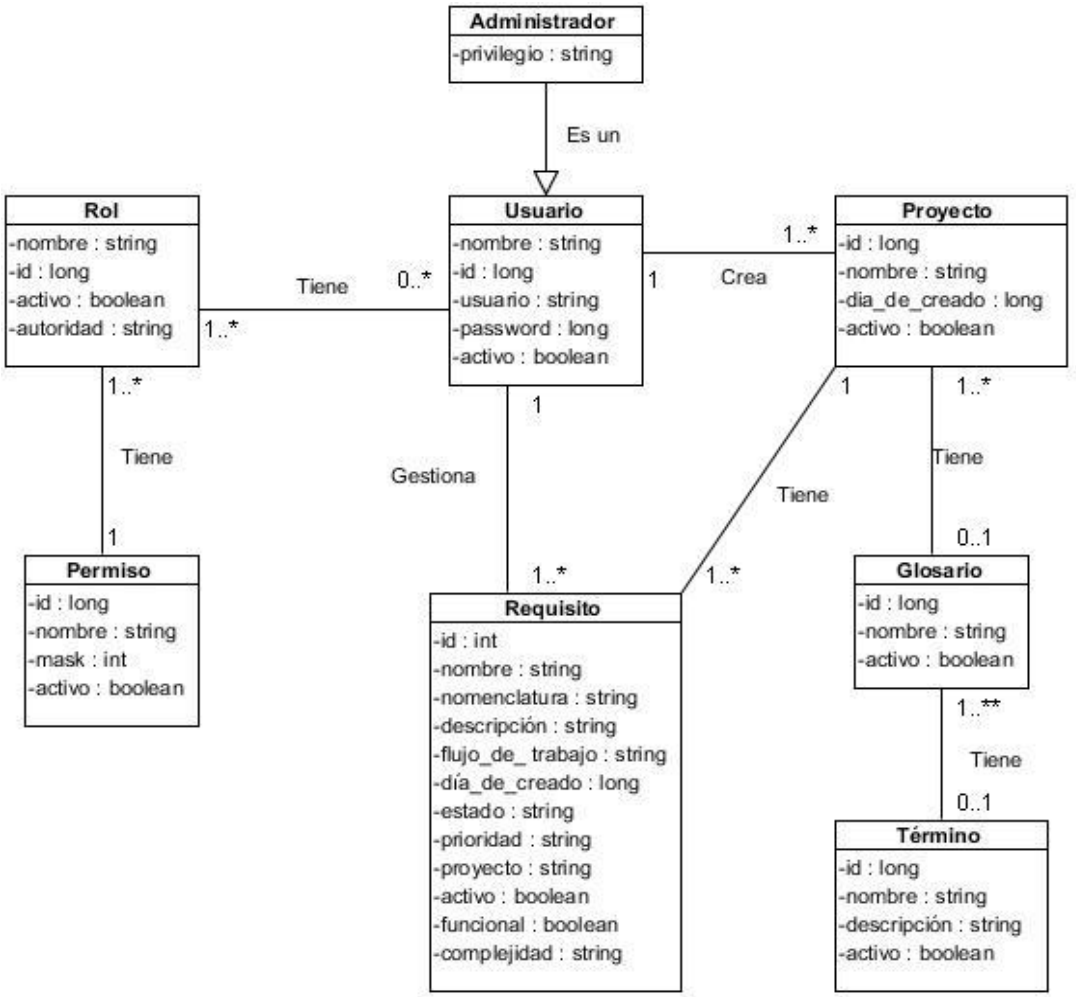

Figura 1. Modelo Conceptual

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 emitir el 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 Requisitos

Para realizar la captura de los requisitos se hizo necesaria la utilización de las técnicas mencionadas en el capítulo anterior como son: la entrevista, la tormenta de ideas y sistemas existentes.

2.4.1 Requisitos Funcionales (RF)

La herramienta a desarrollar debe cumplir con la realización de los siguientes requisitos:

RF1- Seleccionar Proyectos

RF2- Gestionar Proyectos
2.1 Buscar proyectos.
2.2 Adicionar proyecto
2.3 Editar proyecto
2.4 Eliminar proyecto
2.5 Consultar proyectos por grupo.
2.6 Ordenar proyectos.

RF3- Gestionar Requisitos
3.1 Ordenar requisitos.
3.2 Buscar requisitos.
3.3 Calcular peso normalizado.

3.4 Determinar complejidad del requisito.

3.4 Modificar complejidad del requisito.

3.5 Consultar requisitos por su complejidad.

3.6 Mostrar ayuda para determinar complejidad.

RF4- Gestionar Configuración

4.1 Adicionar configuración.

4.2 Editar configuración.

4.3 Eliminar configuración.

4.4 Ordenar datos en la gestión de la configuración.

RF5- Buscar Datos

5.1 Buscar datos en flujo de trabajo.

5.2 Buscar datos en tipo de estado

5.3 Buscar datos en glosario de términos.

5.4 Buscar datos en términos.

5.5 Buscar datos en trazabilidad.

5.6 Buscar datos en usuario.

5.7 Buscar datos en roles.

Las especificaciones de requisitos se encuentran en el repositorio del departamento Soluciones Empresariales, pero a continuación se muestra la del requisito determinar complejidad:

Tabla 5. Especificación del Requisito determinar complejidad

Precondiciones	Se ha registrado al menos un requisito en el sistema.
Flujo de eventos	
Flujo básico	
1	Se introducen los datos de las variables del requisito para el cálculo de la complejidad: Reutilización Facilidad de Cambio Objetivos Específicos de Seguridad Consultas Externas Escalabilidad Interfaces Diferentes Comportamientos Lógica del Negocio

	Restricciones de Validación	
	Dependencia con otros requisitos	
2	El sistema valida (ver validación 1) los datos introducidos.	
3	Si los datos son correctos el sistema los registra.	
4	El sistema confirma el registro de los datos y realiza el cálculo de la complejidad.	
5	Concluye el requisito.	
Pos-condiciones		
1	Se registró en el sistema un nuevo requisito con el cálculo de su complejidad.	
2		
Flujos alternativos		
Flujo alternativo 4.a Información errónea		
1	El sistema señala los datos erróneos y permite corregirlos.	
2	El usuario corrige los datos.	
3	Volver al paso 2 del flujo básico.	
Pos-condiciones		
1	N/A	
Flujo alternativo 4.b Información incompleta		
1	El sistema señala los datos vacíos y permite corregirlos.	
2	El usuario corrige los datos.	
3	Volver al paso 2 del flujo básico.	
Pos-condiciones		
1	N/A	
Flujo alternativo *.a El usuario cancela la acción		
1	Concluye el requisito.	
Pos-condiciones		
1	No se registran los datos.	
Validaciones		
1	Se validan los datos según lo establecido en el Modelo Conceptual 1.0	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	**Requisito Funcional**	Visibles en la interfaz: Reutilización Facilidad de Cambio Objetivos Específicos de Seguridad Consultas Externas Escalabilidad Interfaces Diferentes Comportamientos Lógica del Negocio Restricciones de Validación Dependencia con otros requisitos
Requisitos especiales	N/A	
Asuntos	N/A	

pendientes

2.4.2 Requisitos No Funcionales (RNF)

RNF 1- Requisitos de apariencia o interfaz externa

La herramienta deberá poseer una interfaz web sencilla, amigable, lo más atractiva y clara posible para el usuario.

RNF 2 - Requisitos de uso

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

RNF 3 - Requisitos 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 - Requisitos 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 el 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. Además debe estar instalado el Java Runtime Environment (JRE) versión 1.6 o superior y el Sistema Operativo: Windows XP o superior, o GNU Linux.

RNF 5 - Requisitos 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- Requisitos de restricciones en el diseño y la implementación

Los componentes de la herramienta 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.

2.4.3 Aplicación de técnicas de validación de requisitos

Los requisitos una vez definidos necesitan ser validados. Las técnicas que se utilizaron para la validación de los requisitos identificados fueron las siguientes:

Prototipo de interfaz: una vez terminada la especificación de los requisitos identificados, los mismos fueron entregados a la analista principal para su revisión. Este análisis se realiza con el objetivo de verificar que cada requisito responda a las necesidades del usuario y se reflejen en las interfaces propuestas. Cada plantilla de especificación se revisó para encontrar incoherencias y falta de claridad para poder corregirlos y hacer la descripción de forma más detallada.

Revisión Técnica Formal (RTF) y auditorías: una vez terminadas las especificaciones de los requisitos se realizaron revisiones formales por parte de especialistas, con un control estricto del documento para su primera versión con un total de nueve no conformidades, las cuales fueron arregladas para la segunda revisión que resultó ser satisfactoria. Con este proceso se pudo validar que la interpretación de cada una de las especificaciones no fuera ambigua, y que cada uno de los requisitos cumplía con lo que necesitaba el usuario final.

2.5 Mecanismos de diseño

Los mecanismos de diseño se utilizan con el objetivo de facilitar y abreviar los diagramas de clases. Cada diseñador establece sus propios mecanismos de diseño, teniendo en cuenta los patrones y estilos seleccionados. Para el diseño de la herramienta se utilizaron los mecanismos de diseño para las páginas clientes y controladoras que se utilizan en el sistema SARE.

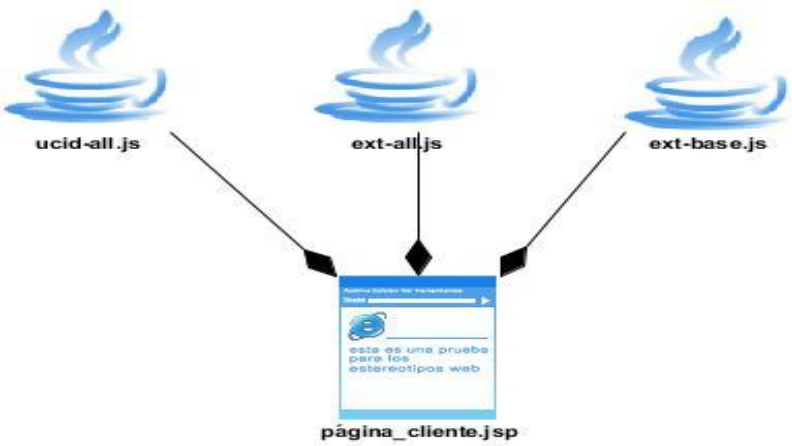

Figura 2. Mecanismos de diseño para clases clientes

ext-all.js: es la encargada de la creación de los componentes visuales de la vista. Está incluida dentro de las clases que trae ExtJS.

ext-base.js: contiene lo que necesita ExtJS para su correcto funcionamiento.

ucid-all.js: encargada de mostrar la interfaz estándar.

2.6 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 además para visualizar las relaciones entre las clases que involucran el sistema. A continuación se muestra en el diagrama de clases del diseño:

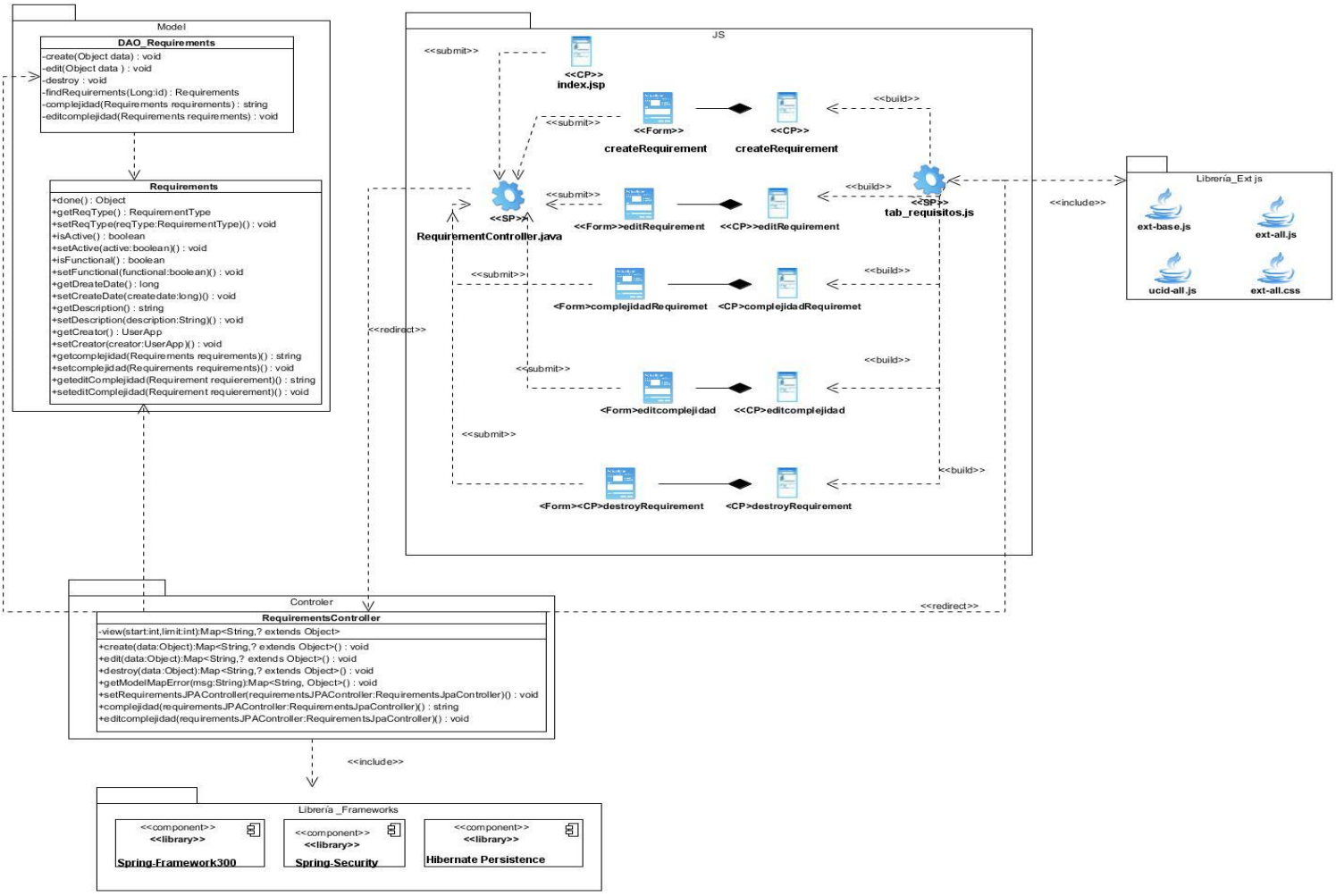

Figura 3. Diagrama de clases del diseño del Gestionar Requisitos

- ✓ **Descripción del diseño de clases para el escenario de Requisitos:**

Tabla 6. Descripción de las clases del diseño del escenario Requisitos

Clases	Descripción
Paquete Controller	Encargado de controlar todos los datos persistentes dentro del componente.
RequirementsController	Clase controladora que sirve de puente entre la vista y el modelo. Contiene los métodos necesarios para adicionar, modificar, eliminar, cálculo de la complejidad, modificar complejidad y buscar requisitos.

Paquete JS	Responsable de visualizar a través de los JS la información necesaria en el escenario Requisitos.
Tab_Requirements	Debe generar de forma dinámica a través de la librería Extjs los componentes necesarios a través de los cuales se adicionen, modifiquen, eliminen, lleven a cabo el cálculo de la complejidad, modifiquen complejidad y busquen los requisitos. Además es la responsable de enviar y recibir los datos de la controladora.
Paquete models	Encargado de manejar los datos persistentes dentro del componente. Contiene el Bussines y el Domain.
DAO_Requirements	Clase del negocio que permite entre otras funcionalidades iniciar la conexión a la base de datos.
Requirements	Clase que lleva a cabo la conexión, manejo y acceso a la base de datos.
Librería Extjs	Contiene los componentes generados a través de la librería Java Script Extjs.
Librería_Framewoks	Contiene los componentes generados a través de las librerías de los frameworks sprint, sprint security e hibernate.

Los restantes diagramas de clases y su especificación se encuentran en el repositorio del departamento Soluciones Empresariales.

2.7 Diagramas de secuencia

Para la realización correcta de los requisitos es factible el empleo de los diagramas de secuencia estos 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 muestra el diagrama de secuencia correspondiente al escenario:

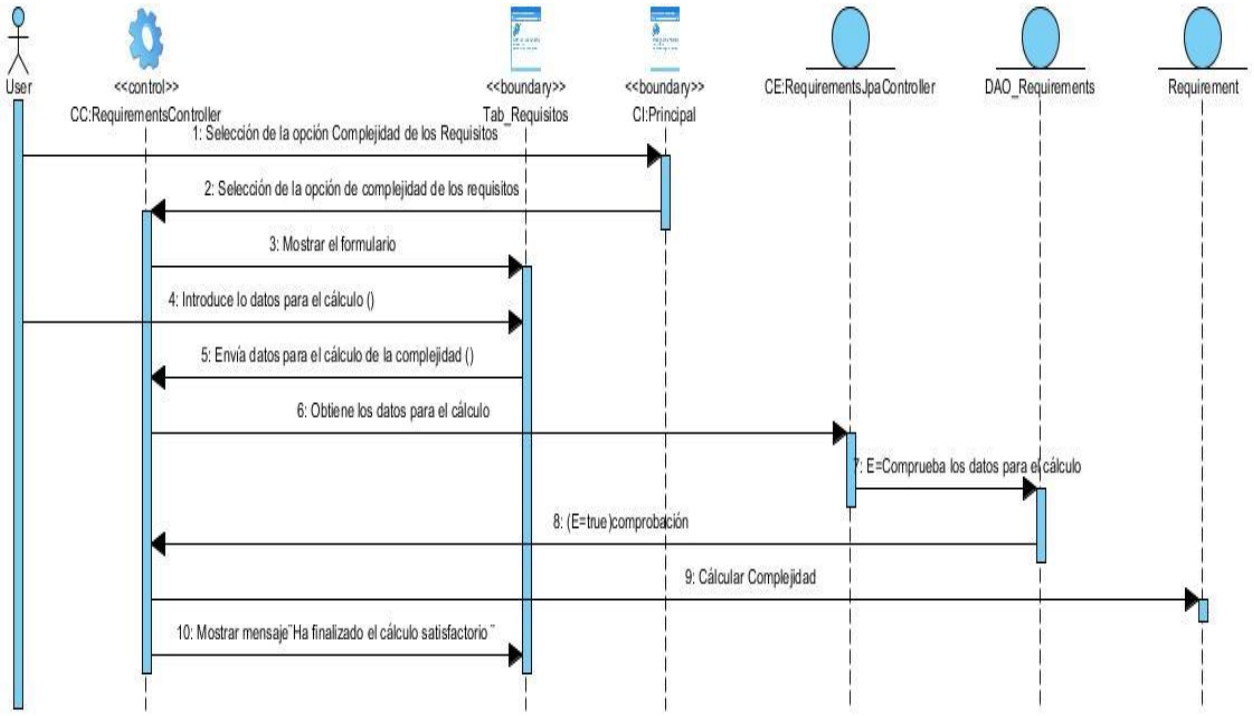

Figura 4. Diagrama de secuencia del Gestionar Requisitos, Escenario: determinación de la complejidad

Los restantes diagramas de secuencia se encuentran en el repositorio del departamento Soluciones Empresariales.

2.8 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. (34)

A continuación se presenta el diagrama de clases persistentes:

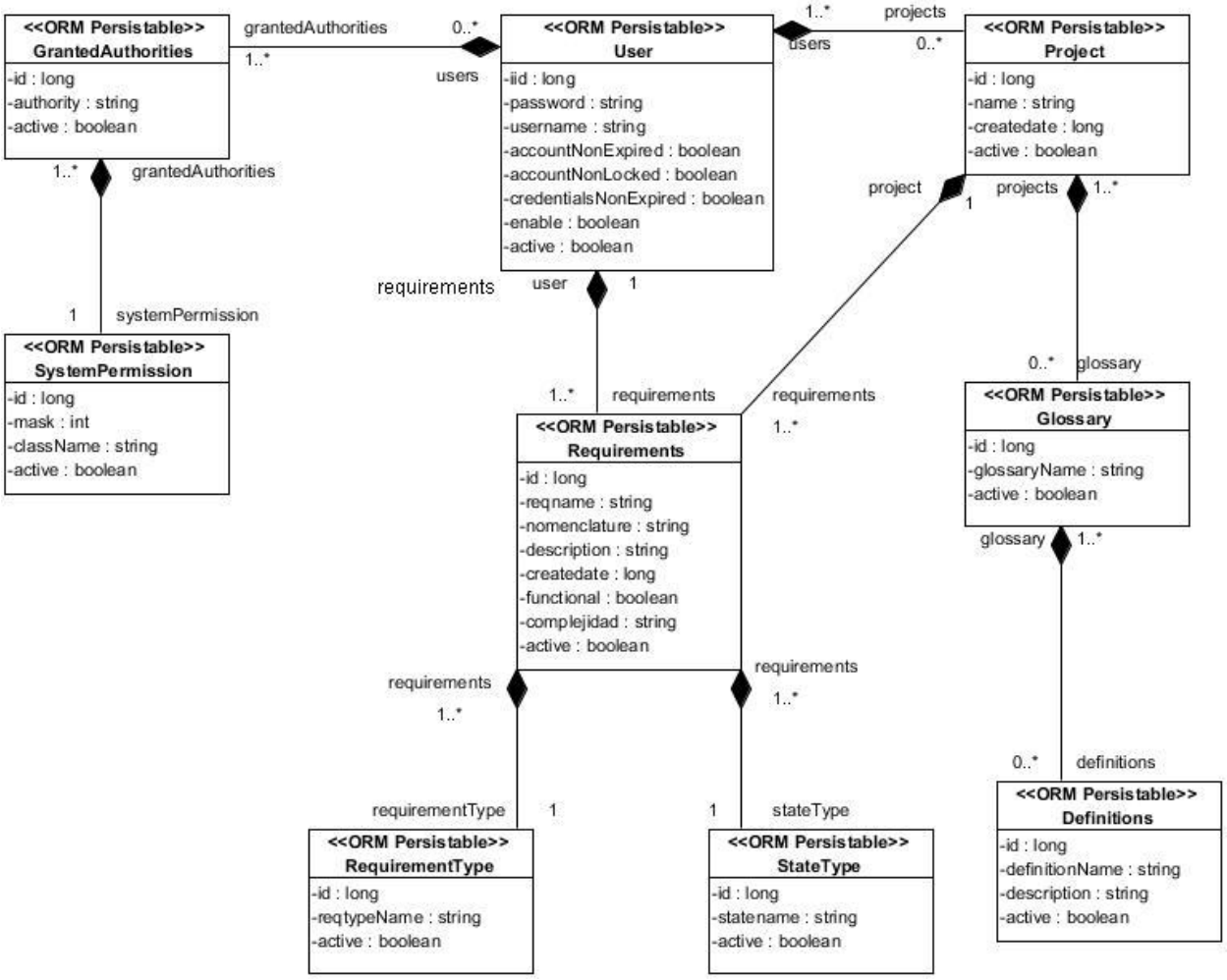

Figura 5. Diagrama de clases persistentes

2.9 Modelo de datos

Un modelo de datos es un lenguaje utilizado para la descripción de una base de datos**.** Por lo general permite describir las estructuras de datos de la base de datos (el tipo de los datos que incluye la base de datos y la forma en que se relacionan), las restricciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base de datos).

A continuación se presenta el modelo de datos de la herramienta el cual cuenta con 13 tablas:

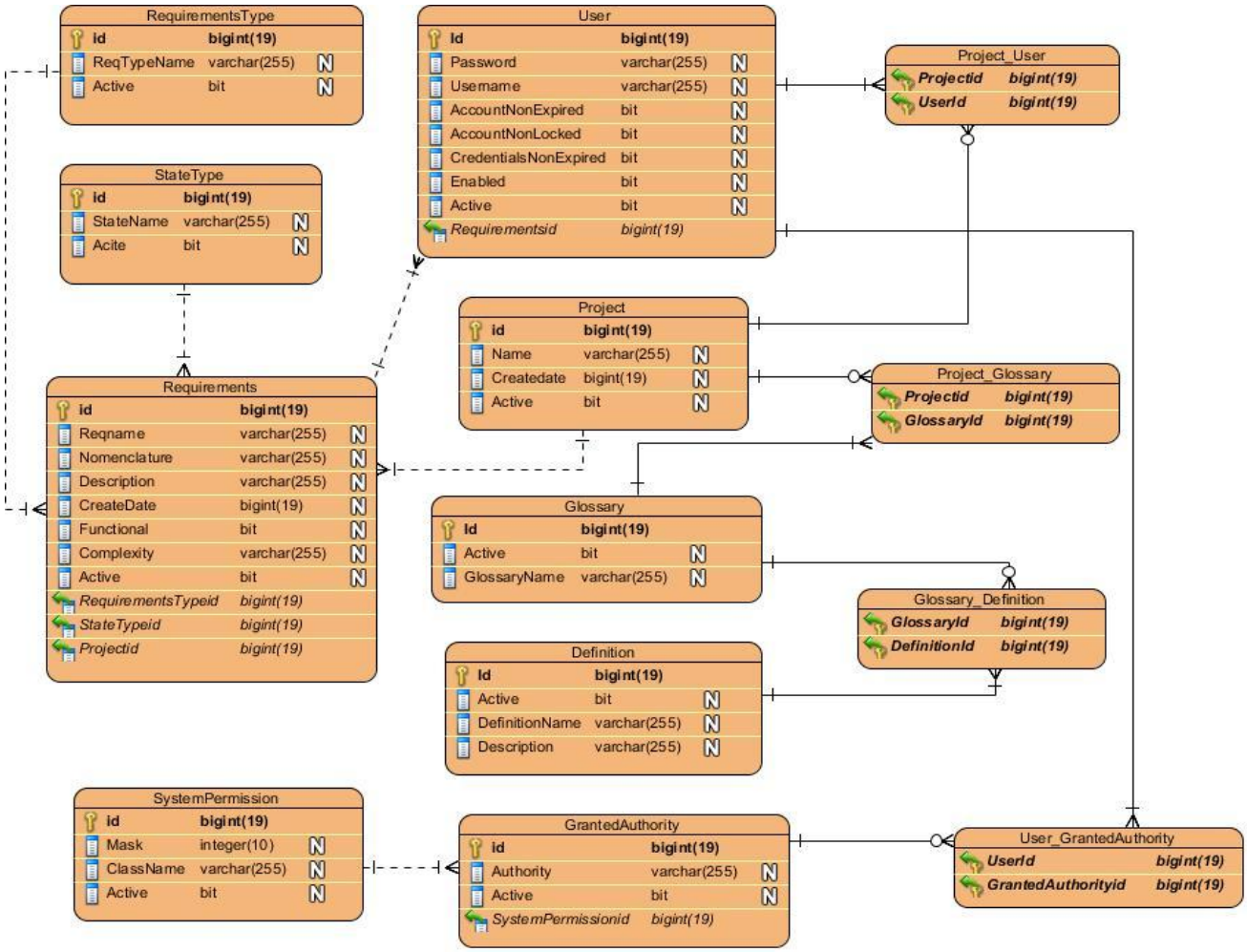

Figura 6. Modelo de datos

2.10 Patrones de diseños utilizados

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. (35)

Estos patrones se evidencian en la herramienta mediante el Bajo Acoplamiento en la clase UserAppJpaController.java, este 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.)

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 una herramienta orientada 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.

2.11 Validación del diseño propuesto

Una métrica es un instrumento que cuantifica un criterio y persigue comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado al nivel del proyecto. Para la evaluación de la calidad del diseño propuesto en la herramienta se hizo un estudio de las métricas básicas inspiradas en la calidad del diseño orientado a objeto, en el mismo se abarcan atributos de calidad que permiten medir la calidad del diseño propuesto. Dentro de estos se encuentran: (36)

Responsabilidad: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad de implementación: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

Reutilización: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Acoplamiento: consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy ligada a la característica de Reutilización.

Complejidad del mantenimiento: consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costos y la planificación del proyecto.

Cantidad de pruebas: consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.

Las métricas concebidas como instrumento para evaluar la calidad del diseño y su relación con los atributos de calidad definidas son las siguientes:

1. **Tamaño Operacional de Clase (TOC):** se refiere al número de métodos pertenecientes a una clase. Está determinada por los atributos: Responsabilidad, Complejidad de implementación y la Reutilización, existiendo una relación directa con los dos primeros e inversa con el último antes mencionado.

Tabla 7. Tamaño operacional de clase (TOC)

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Tabla 8. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC

Atributos	Categoría	Criterio
Responsabilidad	Baja	< =Prom. (7.19)
	Media	Entre Prom. Y 2* Prom
	Alta	> 2* Prom
Complejidad de implementación	Baja	< =Prom
	Media	Entre Prom. y 2* Prom
	Alta	> 2* Prom
Reutilización	Baja	> 2*Prom

	Media	Entre Prom. y 2* Prom
	Alta	<= Prom

Tabla 9. Umbrales para el TOC

Tamaño operacional de la clase	Criterio
Pequeña	<= Prom.(7.19)
Media	Entre Prom. y 2 * Prom.
Grande	> 2* Prom.

Resultados del instrumento de evaluación de la métrica Tamaño Operacional de Clase (TOC)

Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos, ver Figura 7:

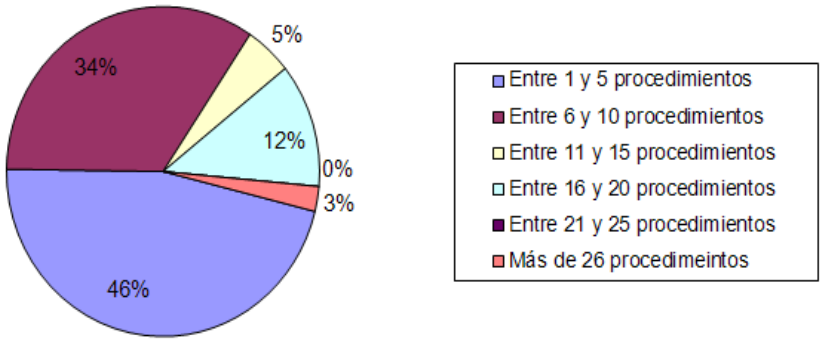

Figura 7. Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad, ver Figura 8:

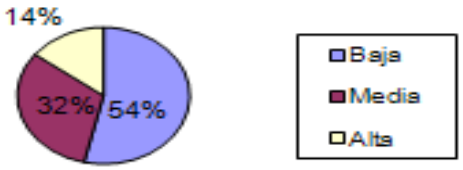

Figura 8. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación, ver Figura 9:

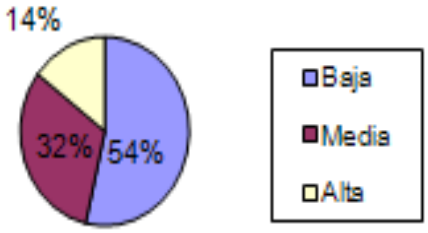

Figura 92. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización, ver Figura0:

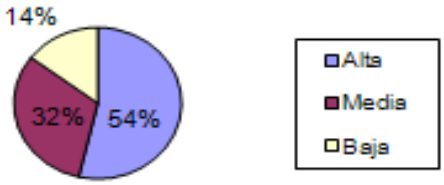

Figura 10. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto para la herramienta es simple y tiene una calidad aceptable, teniendo en cuenta que la mayoría de las clases (45%) posee menos cantidad de

operaciones que la media registrada en las mediciones. Los atributos de calidad se encuentran en un nivel satisfactorio en el 54% de las clases, de manera que se puede observar cómo se fomenta la Reutilización (elemento clave en el proceso de desarrollo de software) y cómo están reducidas en menor grado la Responsabilidad y la Complejidad de implementación.

2. **Relaciones entre Clases (RC):** dado por el número de relaciones de uso de una clase. Está determinada por los atributos: Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas, existiendo una relación directa con los tres primeros e inversa con el último antes mencionado.

Tabla 10. Relaciones entre clases (RC)

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 11. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica RC

Atributos	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

Complejidad de mantenimiento	Baja	< =Prom (2.12)
	Media	Entre Prom. y 2* Prom
	Alta	> 2* Prom
Reutilización	Baja	> 2*Prom
	Media	Entre Prom. y 2* Prom
	Alta	<= Prom
Cantidad de Pruebas	Baja	< =Prom
	Media	Entre Prom. y 2* Prom
	Alta	> 2* Prom

Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC)

Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos, ver Figura 11:

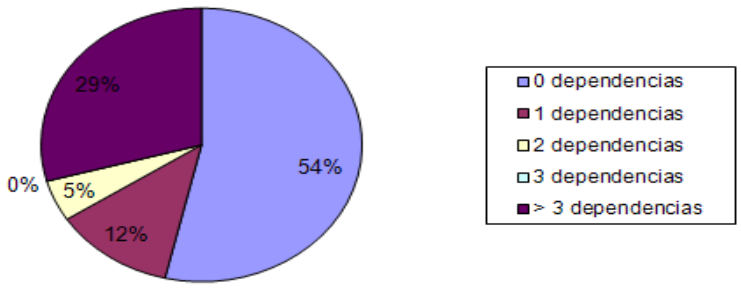

Figura 113. Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento, ver Figura 12:

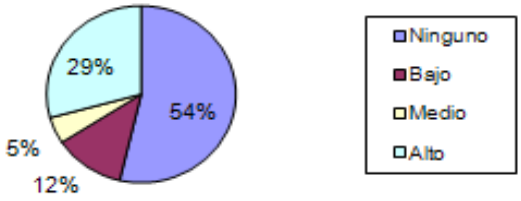

Figura 12. Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento, ver Figura 13:

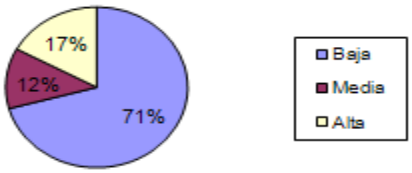

Figura 13. Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de pruebas, ver Figura 14:

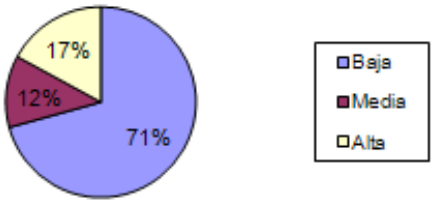

Figura 14. Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de pruebas

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización, ver Figura 15:

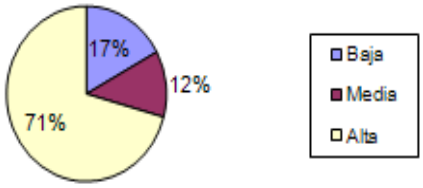

Figura 15. Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica RC, se puede concluir que el diseño propuesto para la herramienta es simple y tiene una calidad aceptable, teniendo en cuenta que la mayoría de las clases (71%) poseen 2 o menos dependencias respecto a otras. Los atributos de calidad se encuentran en un nivel satisfactorio, en el 66% de las clases el grado de acoplamiento es mínimo, la Complejidad de mantenimiento, la Cantidad de pruebas y la Reutilización se comportan favorablemente para un 71% de las clases.

Conclusiones parciales

Los artefactos generados en este capítulo son la base para el desarrollo de la herramienta a realizar en el sistema SARE como son:

- Se definieron todos los requisitos de la herramienta, los cuales cumplen con todas las funcionalidades requeridas para realizar el cálculo de la complejidad de los requisitos funcionales del software.
- Se describieron los patrones de diseño utilizados durante el desarrollo, los cuales proporcionaron una baja dependencia y una alta reutilización entre las clases de la misma.
- Se realizó el modelo de datos de la herramienta con el objetivo de que este sea capaz de almacenar toda la información referente a la complejidad de los requisitos funcionales del software, mostrándose además las relaciones existentes entre las diferentes tablas de la base de datos.
- Se realizaron y describieron los diagramas de clases de diseño del escenario Requisitos para alcanzar un entendimiento mejor y más profundo de cómo se realizará la implementación de la herramienta.

- Fueron expuestas las métricas TOC y RC para validar el diseño, las cuales arrojaron resultados satisfactorios sobre este, demostrando que era simple y que los atributos de calidad alcanzaban niveles favorables.

Capítulo 3: Implementación y Prueba

Introducción

En este capítulo se realiza una breve descripción del modelo de implementación, así como los estándares a utilizar durante el desarrollo de la herramienta. También se describen las clases y funcionalidades, además de realizar el diagrama de componentes y de despliegue para obtener una visión más clara sobre la misma. Por último se describirán las pruebas de caja negra y caja blanca realizadas al software y sus resultados.

3.1 Modelo de implementación

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

3.1.1 Diagrama de componentes

Dentro del modelo de implementación se encuentran los diagramas de componentes. Un diagrama de representa cómo un sistema de software es dividido en componentes, mostrando además las dependencias entre estos. Este tipo de diagrama contiene componentes, interfaces y sus relaciones, puede contener también paquetes que se utilizan para agrupar elementos del modelo (37). Teniendo en cuenta el modelo de desarrollo a utilizar se define como estructura básica a los componentes.

En la Figura 16 se muestra el diagrama de componentes de la herramienta:

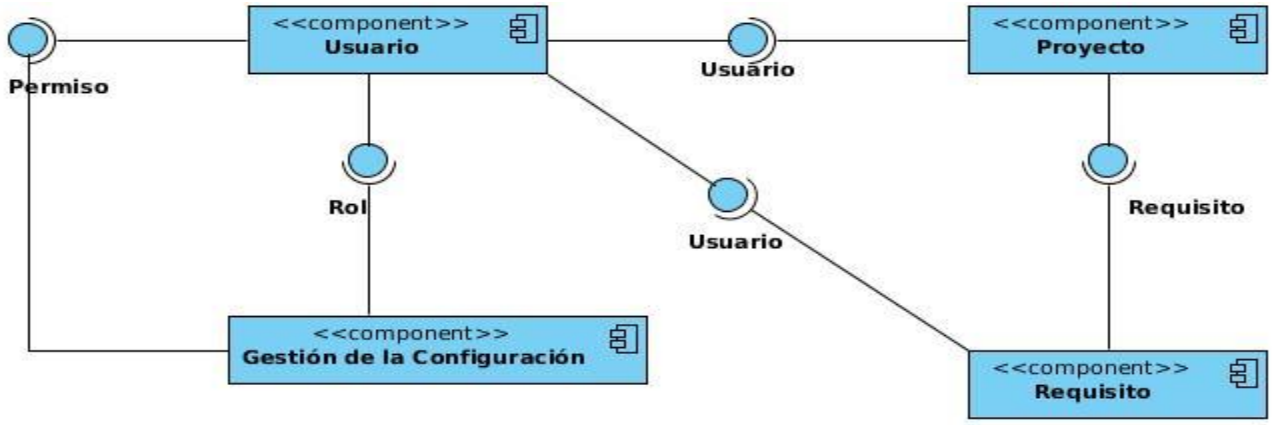

Figura 16. Diagrama de componente

A continuación, una explicación más detallada de la funcionalidad de cada uno de estos componentes a partir de los servicios que brindan y reciben:

Usuario: se encarga de gestionar los usuarios, además según el rol que este posea registra los proyectos y gestionar los requisitos emitiendo el seguimiento dentro de la herramienta. Proporciona la interfaz usuario a los componentes requisito y proyecto y la interfaz rol al componente gestión de la configuración.

Proyecto: el componente depende de la interfaz usuario y se encarga de gestionar los proyectos.

Requisito: este componente se encarga de realizar la gestión de los requisitos a través de los usuarios y permite saber a qué proyecto pertenece cada requisito dependiendo además de la interfaz usuario.

Gestión de la Configuración: el componente depende de la interfaz rol y brinda permisos a los usuarios tales como: lectura, edición y eliminación para lograr una mayor satisfacción de los mismos dentro de la herramienta proporcionando la interfaz permiso al componente usuario.

3.1.2 Diagrama de despliegue

El diagrama de despliegue muestra la configuración de los nodos de procesamiento en tiempo de ejecución, los vínculos de comunicación entre ellos, y las instancias de los componentes y objetos que residen en ellos. Está compuesto por nodos, dispositivos y conectores. El propósito del modelo de despliegue es capturar la configuración de los elementos de procesamiento y las

conexiones entre estos elementos en el sistema. Permite el mapeo de procesos dentro de los nodos, asegurando la distribución del comportamiento a través de aquellos nodos que son representados (38). Se definirá una arquitectura cliente-servidor detallada de la siguiente manera, ver Figura 17.

A continuación se muestra el diagrama de despliegue de la herramienta:

Figura 17. Diagrama de despliegue

PC_Cliente: computadora en la cual la herramienta se ejecutará a través de un navegador que soporta Java Script y Adobe Reader 5.0 o superior, en este caso se debe usar el Mozilla Firefox.

Servidor Web: radica la lógica de negocio de la herramienta. Servidor Web Apache Tomcat 6.0, utilizando el lenguaje de programación java.

Servidor de Base de Datos: servidor de Datos PostgreSQL 9.1, donde se encuentra la base de datos que utiliza la herramienta, este puede estar instalado en la misma computadora donde se encuentra el servidor Web.

El nodo PC_Cliente estará conectado mediante el Protocolo de Transferencia de Hipertexto HTTPS por el puerto 443 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.

3.2 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. (39)

3.2.1 Estándares de codificación

Las convenciones o estándares de codificación son un conjunto de directrices que especifican cómo debe escribirse el código fuente. Un estándar de código se basa en la estructura y apariencia física de un programa, con el fin de facilitar la lectura, comprensión, mantenimiento del código y reutilización a lo largo del proceso de desarrollo de un software y no en la lógica del programa. Este no solo busca definir la nomenclatura de las variables, objetos, métodos y funciones, sino que también tiene que ver con el orden y la legibilidad del código, aspecto crucial a la hora de darle mantenimiento y mejorar las funcionalidades de un software. (40)

Es válido argumentar que para los estándares de codificación de la herramienta para determinar el cálculo de la complejidad de los requisitos funcionales de software se siguen las mismas políticas que están definidas en SARE y se muestran a continuación:

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 identificación 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.

3.2.2 Ejemplo de código fuente

La mayoría de los métodos implementados responden a funciones básicas de inserción, actualización, eliminación, cálculo de la complejidad de los requisitos y listar los requisitos por sus funcionalidades en la herramienta. Así de esta forma se genera la mayor parte del código de acceso a los datos. A continuación se muestra en la figura 18 el código del método checkPermission (Requirements requirements, long mask, String className) y se ofrece una breve descripción de este.

```
private boolean checkPermission(Requirements requirements, long mask, String className){
    if(GrantsChecker.checkGrants(mask, "ToRequirement")){
        return true;
    }
    else{
        Collection<Restriction> permittedProjects = GrantsChecker.getObjectsPermitted(mask, className)
        for (Restriction restriction : permittedProjects) {
            String idToCheck = "";

            if(className.equals("Project")){
                idToCheck = requirements.getProject().getId().toString();
            }
            else
            {
                if(className.equals("RequirementType")){
                    idToCheck = requirements.getReqType().getId().toString();
                }
            }

            if(restriction.getField().equals(idToCheck)){
                return true;
            }
        }
    }
    // Validar si posee permisos suficientes para crear un requisito en ese proyecto
    return false;
}
```

Figura 18. Ejemplo de código fuente: Método adicionar requisito

Este método se encarga de chequear los permisos que debe poseer un requisito en el proyecto creado, a través del usuario autenticado en la herramienta.

3.3 Pruebas de software

Las pruebas de software son una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de un programa u ordenador, probando el comportamiento del mismo. Se considera una prueba exitosa, si se demuestran deficiencias en el software. Las fallas pueden ser en el código o en el modelado, en dependencia del tipo de pruebas que se le apliquen al software. La prueba no puede asegurar la ausencia de defectos; solo puede demostrar que existen defectos en el software. (35)

3.3.1 Pruebas de caja blanca

La prueba de caja blanca también se conoce como prueba de caja transparente o de cristal. Esta prueba consiste específicamente en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento. (35)
Dentro de la prueba de caja blanca se incluyen las Técnicas de Pruebas que serán descritas a continuación:

- **Prueba del Camino Básico:** permite obtener una medida de la complejidad lógica de un diseño y usar la misma como guía para la definición de un conjunto de caminos básicos.
- **Prueba de Condición:** ejercita las condiciones lógicas contenidas en el módulo de un programa. Garantiza la ejecución por lo menos una vez de todos los caminos independientes de cada módulo, programa o método.
- **Prueba de Flujo de Datos:** se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa. Garantiza que se ejerciten las estructuras internas de datos para asegurar su validez.
- **Prueba de Bucles:** se centra exclusivamente en la validez de las construcciones de bucles. Garantiza la ejecución de todos los bucles en sus límites operacionales.

La prueba de caja blanca empleada en la solución desarrollada fue la Prueba del Camino Básico, la cual permite obtener una medida de la complejidad lógica del diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución, garantizando con estos que durante la prueba se ejecute por lo menos una vez cada sentencia del programa. (35)

Para realizar esta técnica es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar. A continuación se enumeran las sentencias de código del procedimiento realizado sobre el método editComplexity(String complexity, Long id) de la clase RequirementsController.

```
public void editComplejidad(String complejidad, long id) throws NonexistentEntityException, Exception { //1
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("TrackingRequirementsPU"); //1
    long mask = 2L;//1
    EntityManager em = null; // 1
    Requirements requirements = null; //1
    try { //2
        requirements = findRequirements(id);// 3
        requirements.setComplejidad(complejidad);//3
    }  //4
    catch (Exception ex) { //5
        requirements = new Requirements();//6
    }//7
    boolean flag = checkPermission(requirements, mask, "Project"); // 8
    if (!flag) { //9
        throw new AccessForbiddenException("No tiene permiso para editar requisitos"); //10
    } //11
    else { // 12
        flag = checkPermission(requirements, mask, "RequirementType"); //13
        if (!flag) { //14
            throw new AccessForbiddenException("No tiene permiso para crear requisitos"); //15
        } //16
        else { //17
            try { //18
                em = emf.createEntityManager(); //19
                em.getTransaction().begin(); //19
                requirements = em.merge(requirements); //19
                em.getTransaction().commit(); //19
            } //20
            catch (Exception ex) { //21
                String msg = ex.getLocalizedMessage(); //22
                if (msg == null || msg.length() == 0) { //23
                    if (findRequirements(id) == null) {0000000000 // 24
                        throw new NonexistentEntityException("The requirements with id " + id + " no longer exi
                        } //26
                    } //27
                    throw ex; //28
                } //29
                finally { // 30
                    if (em != null) { //31
                        em.close(); //32
                        emf.close(); //32
                    } //33

                } //34
            }  //35
        }  //36
} //37
```

Figura 19. Código del Método editComplejidad (String complejidad, Long id)

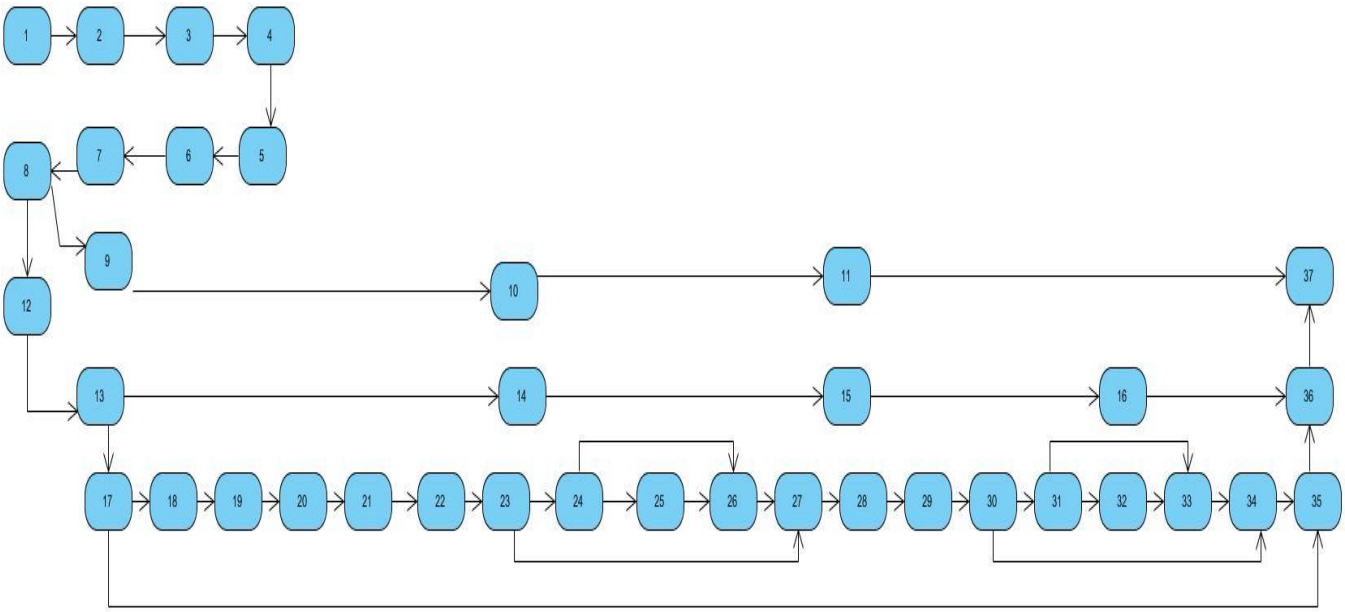

Figura 20. Grafo de flujo asociado al algoritmo editComplejidad (String complejidad, Long id)

Fórmulas para calcular la complejidad ciclomática:

V (G) = (A - N) + 2

Donde **"A"** es la cantidad de aristas y **"N"** la cantidad de nodos.

V (G) = (43 - 37) + 2

V (G) = 8

V (G) = P + 1

Siendo **"P"** la cantidad de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 7 + 1

V (G) = 8

V (G) = R

Donde **"R"** representa la cantidad de regiones en el grafo.

V (G) = 8

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es de 8, lo que significa que existen ocho posibles caminos

por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Tabla 12. Caminos básicos del flujo

Número	Caminos básicos
1	1-2-3-4-5-6-7-8-9-10-11-37
2	1-2-3-4-5-6-7-8-12-13-14-15-16-36-37
3	1-2-3-4-5-6-7-8-12-13-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37
4	1-2-3-4-5-6-7-8-12-13-17-18-19-20-21-22-23-24-26-27-28-29-30-31-32-33-34-35-36-37
5	1-2-3-4-5-6-7-8-12-13-17-18-19-20-21-22-23-27-28-29-30-31-32-33-34-35-36-37
6	1-2-3-4-5-6-7-8-12-13-17-18-19-20-21-22-23-24-25-26-27-28-29-30-34-35-36-37
7	1-2-3-4-5-6-7-8-12-13-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-33-34-35-36-37
8	1-2-3-4-5-6-7-8-12-13-17-35-36-37

Posteriormente de haber determinado los caminos básicos se procede a ejecutar los casos de pruebas para cada uno de estos. Para definir los casos de prueba es necesario tener en cuenta:

- **Descripción:** se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.
- **Condición de ejecución:** se especifica cada parámetro para que cumpla una condición deseada y así ver el funcionamiento del procedimiento.
- **Entrada:** se muestran los parámetros que serán la entrada al procedimiento.
- **Resultado esperado:** se expone el resultado esperado que debe devolver el procedimiento después de efectuado el caso de prueba.

Caso de prueba para el camino básico # 1.

Descripción: los permisos asignados al proyecto no son los suficientes para acceder sobre el requisito seleccionado y no permite realizar dicha operación.
Condición de ejecución: el proyecto tiene permiso sobre el requisito.

Entrada:

- complejidad=¨Alta¨, representa el valor de la complejidad del requisito seleccionado.
- id= 15, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que el proyecto no tiene permisos sobre el requisito.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 2.

Descripción: el proyecto no tiene privilegios suficientes para acceder sobre el requisito seleccionado y no permite crear más requisitos.

Condición de ejecución: el proyecto tiene permiso sobre el requisito.

Entrada:

- complejidad=¨Baja¨, representa el valor de la complejidad del requisito seleccionado.
- id= 10, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que el proyecto no tiene permiso para crear requisitos.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 3.

Descripción: el proyecto tiene privilegios suficientes para acceder sobre los requisitos, lo que este requisito no se encuentra.

Condición de ejecución: exista el requisito en la base de datos y se muestre en el sistema.

Entrada:

- complejidad=¨Alta¨, representa el valor de la complejidad del requisito seleccionado.
- id= ¨ ¨, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que el requisito no se encuentra en el proyecto.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 4.

Descripción: el proyecto tiene privilegios suficientes para acceder sobre el requisito seleccionado y permite crear más requisitos.

Condición de ejecución: el proyecto tiene permiso sobre el requisito.

Entrada:

- complejidad=¨Media¨, representa el valor de la complejidad del requisito seleccionado.
- id= 2, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que el requisito fue actualizado satisfactoriamente.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 5.

Descripción: el proyecto tiene privilegios suficientes para acceder sobre el requisito seleccionado y permite crear más requisitos.

Condición de ejecución: que el sistema valide los datos para mostrar el mensaje.

Entrada:

- complejidad=¨Media¨, representa el valor de la complejidad del requisito seleccionado.
- id=2, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que los datos de entrada son vacíos, inválidos o que ya se encuentren en el sistema.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 6.

Descripción: el proyecto tiene privilegios suficientes para acceder sobre el requisito seleccionado no puede encontrar el requisito.

Condición de ejecución: que el sistema valide los datos y encuentre el requisito**.**

Entrada:

- complejidad=¨Alta¨, representa el valor de la complejidad del requisito seleccionado.
- id=22**,** representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que no se pudo encontrar el requisito.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 7.

Descripción: el proyecto tiene privilegios suficientes para acceder sobre el requisito seleccionado y permite crear más requisitos.

Condición de ejecución: que el sistema valide los datos para mostrar el mensaje.

Entrada:

- complejidad=¨Media¨, representa el valor de la complejidad del requisito seleccionado.
- id=2, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que los datos de entrada son vacíos, inválidos o que ya se encuentren en el sistema.

Resultado obtenido: satisfactorio.

Caso de prueba para el camino básico # 8.

Descripción: el proyecto tiene privilegios suficientes para acceder sobre el tipo de requisito seleccionado.

Condición de ejecución: el proyecto tiene permiso sobre el tipo requisito.

Entrada:

- complejidad=¨Baja¨, representa el valor de la complejidad del requisito seleccionado.
- id=20, representa el identificador del requisito.

Resultado esperado: se espera que muestre un mensaje informando que el proyecto no tiene suficiente privilegios sobre el tipo de requisito.

Resultado obtenido: satisfactorio.

3.3.2 Pruebas de caja negra

A este tipo de prueba también se le conoce como prueba de caja opaca o inducida por los datos. Se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación, esta prueba se limita a brindar solo datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo internamente, es decir, solo trabaja sobre su interfaz externa .(35)

En esencia permite encontrar:(35)

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.

Para cada uno de los requisitos funcionales fueron diseñados los casos de prueba que se encuentran en el repositorio del departamento Soluciones Empresariales. A continuación se especifica el caso de prueba del requisito determinar complejidad de los requisitos funcionales.

Caso de prueba para el requisito: Determinar la complejidad de los requisitos funcionales.

Condiciones de ejecución

- El usuario se debe autenticar en el sistema, además debe tener los permisos para ejecutar esta acción.
- Se selecciona un proyecto y debe estar insertado un requisito que pertenezca a este proyecto. A partir de esa acción se debe seleccionar la opción del menú: **Requisitos /Complejidad de los requisitos.**

Tabla 13. Caso de prueba para el requisito determinar complejidad

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1. Cálculo de la complejidad de los requisitos.	Debe calcular la complejidad de los requisitos en la herramienta.	EP 1.1: Calcular complejidad introduciendo datos válidos.	– Se introducen los datos de las variables para el cálculo correctamente. – Se presiona el botón **Calcular**. – Se muestra un mensaje de información "*Se ha calculado la complejidad satisfactoriamente*" – Se presiona el botón **Aceptar**.
		EP 1.2: Calcular complejidad introduciendo datos inválidos.	– Se introducen los datos inválidos en la herramienta. – Se presiona el botón **Calcular**. – Se muestra un mensaje de error "*Se han introducido datos inválidos*" – Se presiona el botón **Aceptar**.
		EP 1.3: Calcular complejidad a un requisito que ya tiene realizado el cálculo de la complejidad.	– Se muestra un mensaje de error "*Este requisito ya tiene calculada su complejidad*" – Se presiona el botón **Aceptar**.
		EP 1.4: Calcular complejidad dejando campos vacíos.	– Se introducen los datos dejando campos vacíos. – Se presiona el botón **Calcular**. – Se muestra un mensaje informando del

		error "*No se pueden dejar campos vacíos*" – Se presiona el botón **Aceptar**.
	EP 1.5: Cancelar.	– No se introducen los datos para el cálculo de la complejidad. – Se presiona el botón **Cancelar**.

La herramienta fue sometida a dos iteraciones de pruebas de caja negra. A continuación se muestran los resultados obtenidos:

Tabla 14. Resultados de las iteraciones

No Conformidades		Iteraciones		Total	
		1	2		
Aplicación	Funcionalidad	1	0	29	0
	Interfaz	12			
	Validación	16			
Documentación	Ortografía	1	0	6	0
	Redacción	2			
	Correspondencia	3			

Todas las no conformidades fueron resueltas luego su detección para la primera iteración, permitiendo que la herramienta pasara a una segunda iteración obteniéndose como resultados una herramienta libre de no conformidades.

La herramienta desarrollada fue presentada ante el cliente, mostrándose satisfecho con el producto y emitiendo su acta de aceptación que avala el cumplimiento de sus necesidades y su total satisfacción. (Ver Anexo 2)

Conclusiones parciales

En este capítulo se realizó la implementación y validación de la herramienta, lo que permitió obtener importantes resultados como:

- Se realizó el diagrama de componentes para mostrar las dependencias lógicas que existen entre ellos y el diagrama de despliegue para visualizar los componentes de software en los nodos físicos de la herramienta.

- Se utilizaron los estándares de implementación a utilizar para lograr un mejor entendimiento del código por los programadores y el futuro mantenimiento a la herramienta.
- Se le realizaron pruebas de caja blanca y caja negra, mostrando resultados satisfactorios, demostrándose la calidad de la herramienta realizada.

Conclusiones Generales

Con la culminación del trabajo de diploma se cumplieron los objetivos propuestos arribando a las siguientes conclusiones:

- Con estudio de diferentes herramientas que gestionan requisitos se pudo concluir que estas presentan ventajas y desventajas de acuerdo a los parámetros medidos, siendo el sistema SARE el único con pasos de avances e indicadores medibles definidos para el cálculo de complejidad de los requisitos funcionales del software.
- Se realizó el análisis y diseño de la herramienta para la determinación de la complejidad de los requisitos funcionales de software, permitiendo recoger todas las necesidades de los clientes, además de lograr un mejor entendimiento y comprensión las funcionalidades a informatizar.
- La solución propuesta fue implementada 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 validó el diseño aplicando métricas donde los resultados arrojados demostraron la presencia de valores positivos en los indicadores medidos.
- La herramienta realizada fue expuesta a diferentes pruebas, las cuales arrojaron resultados satisfactorios demostrando la calidad de la misma.

Recomendaciones

Teniendo como base los resultados de este trabajo y la experiencia adquirida durante el desarrollo del mismo, se recomienda:

- Integrar la herramienta del presente trabajo de diploma con el generador dinámico de reportes.
- Realizar pruebas unitarias y pruebas de rendimiento a la herramienta.
- Configurar la herramienta para que permita gestionar nuevas variables en cualquier entidad de desarrollo de software.

Referencias Bibliográficas

1. Ramos González Leidy. Herramienta para la Gestión de los Requisitos en los proyectos productivos de la Facultad 10. Habana : s.n., 2008.

2. Toro, Amador Duran. Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información. [En línea] 2000. [Citado el: 2 de Octubre de 2012.]

3. Terminology, IEEE Standard Glossary of Software Engineering.[En línea] 24 de Mayo de 2006. [Citado el: 1 de octubre de 2012.] http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html. 610.12-1990.

4. García Mirabal Tatiana. Método para calcular la complejidad de los requisitos funcionales del software de los proyectos productivos del Centro de Informatización de la Gestión de Entidades : s.n., 2012.

5. Soto Lauro Prof. Especificaciones de Requerimientos, Ensenada BC,México Mitecnologico.com. [En línea] [Citado el: 2 de Octubre de 2012.] http://www.mitecnologico.com/Main/EspecificacionesDeRequerimientos..

6. Pérez., María de Lourdes. Ingeniería de Requerimientos. [En línea] 2008. [Citado el: 2 de Octubre de 2012.] http://dgsa.uaeh.edu.mx:8080/bibliotecadigital/bitstream/231104/415/1/Ingenieria%20de%20requerimientos.pdf.

7. Chaves Arias Michael. La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software. [En línea] 2006. [Citado el: 2 de Octubre de 2012.]

8. González Obregón William. Modelo de desarrollo de Software. [En línea] 2012. [Citado el: 5 de Octubre de 2012.]

9. Jacobson, Ivar, Boch, Grady, James. El Proceso Unificado de Desarrollo de Software.Lenguaje UML 2000. pág. 13.

10. Sommerville, Ian. 2002. Ingeniería de Software. Sexta edición. Addison-Wesley : s.n., 2002. ISBN: 970-26-0206-8.

11. Loureiro, Tania Teresa. Análisis y diseño de la solución informática para el subsistema de Caja, del Sistema de Gestión Empresarial Cedrux. 2009.

12. Visual Paradigm. 2007. Visual Paradigm. Free Download Manager.org. [En línea] 2007. [Citado el: 10 de Octubre de 2012.]

http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p.

13. Rumbaugh, James, Ivar .El Lenguaje Modificado de Modelado. 2000. pág. 19.

14. García Javier, Rodríguez José Ignacio,Migno Iñigo,RImaz Aito,Brazáles Alfonso, Larzabal Alberto, Calleja Jesús, García Jon. San Sebastián .*Aprenda JAVA* como si estuviera en primero. s.n:2000.

15.EcuRed: Enciclopedia cubana. [En línea] [Citado el: 15 de Octubre de 2012.] http://www.ecured.cu/index.php/Lenguaje_de_programaci%C3%B3n_Java.

16. Garcia, Yunesky del Río. Implementación del Módulo Índice de Peligrosidad Pre-Delictiva del Proyecto Sistema de Gestión Fiscal. Cuidad de la Habana: s.n., 2010.

17.Tecnología Java.Área de java - Ciberaula,. [En línea] [Citado el: 21 de Octubre de 2012.] http://java.ciberaula.com/articulo/tecnologia_java/

18.Scribd.Java orientado a objetos. [En línea] 2010 .[Citado el: 22 de octubre de 2012.] http://es.scribd.com/doc/10987685/Java-orientado-a-objetos

19. Alvarez, Miguel Angel. Lenguaje de programación de propósito general, orientado a objetos, que también puede utilizarse para el desarrollo web. 2009.

20. Sprint Framework.Capítulo 6.[En línea] [Citado el: 25 de Octubre de 2012.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo6.pdf

21.Spring Framework. Spring Source.org. [En línea] [Citado el: 28 de Octubre de 2012.] http://static.springsource.org/spring-security/site/

22.Ayuda de Extjs3.3.0 [En Línea] [Citado: el 25 Octubre de 2012.] http://dev.sencha.com/deploy/ext-3.3.1/examples/spinner/spinner.html

23. Rosés, Albiol Frances. Introducción a Hibernate. 2009.

24. García, Joaquín. Ingeniero Software. [En línea] 2005. [Citado el: 3 de Octubre de 2012.] http://www.ingenierosoftware.com/analisisydiseno/uml.php.

25.Netbeans 6.9. 2010.[En línea] 2010.[Citado el: 4 de Octubre de 2012.] http://www.desarrolloweb.com/actualidad/netbeans-6-9-final-liberado-3613.html

26. iText Software Corp. PROGRAMAS-GRATIS.NET. [En línea] [Citado el: 29 de Octubre de 2012.] http://itext.programas-gratis.net/.

27. Martínez, Rafael. 2009. PostgreSQL-es. Procedimientos almacenados y PL/pgSQL. [En línea] 2009. [Citado el: 30 de Octubre de 2012.] http://www.postgresql.org.es/node/297.

28. González, Carlos D. Curso PostgreSQL, SQL avanzado y PHP. [En línea] Noviembre de 2012. http://www.usabilidadweb.com.ar/postgre.php.

29. Sencha. Ayuda de Extjs.[En línea] [Citado el: 1 de Noviembre de 2012.] http://www.sencha.com/products/extjs/license/.

30. Palacios Vázquez, Rodolfo. Scribd. [En línea] 2 de Septiembre de 2010. [Citado el: 3 de Noviembre de 2012.] http://es.scribd.com/doc/37957736/Mozilla-Firefox-Original.

31. Machado Machado Armando. Análisis y Diseño de los procesos Administración Compra y Venta para el Sistema de Administración de Relaciones con el Cliente.Habana : s.n., 2012.

32. Lago Ramiro. Patrones de diseño. 2007.

33. Gestión de la Calidad y Pruebas de Sotfware. [En línea] [Citado el: 5 de Noviembre de 2012.] http://pruebasdesoftware.com/laspruebasdesoftware.htm.

34. Quintero JB, de Páez RA, Marín JC. Revista Universidad ,2012.
[Citado: febrero 4, 2013].http://publicaciones.eafit.edu.co/index.php/revista-universidad-eafit/article/view/838

35. Larman, Graig. 2004. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. UML y Patrones. 2004.

36. Pressman, Roger S. Ingeniería de Software, Un enfoque práctico. 2005.

37. Mamani Nina Dania, García Quispe Daniela, Melania Sotes Carrillo. Mayra. Artefacto: Diagrama de componentes. La Paz, Bolivia: s.n., 2009.

38. EcuRed: Enciclopedia cubana. [En línea] [Citado el: 22 de Febrero de 2013.] http://www.ecured.cu/index.php/Diagrama_de_despliegue.

39. Guglielmetti Marcos, Lanzillotta Analía, Alsina Guillem, Yanover David. Master Magazine. Master Magazine. [En lllínea] febrero 11, 2005. [Citado: febrero 24, 2013.] http://www.mastermagazine.info/termino/4328.php.

40. Herranz, Raúl. Utópica Informática. Utópica Informática. [En línea] diciembre 23, 2010. [Citado: febrero 24, 2013.]
http://utopicainformatica.blogspot.com/2010/12/convencionesestandares-de-codificacion.html.

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 Octubre de 2012.] http://contenidodigital.wordpress.com/2008/09/19/hibernate-una-herramienta-de-mapping-ii/.

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

4. Landazuri, Bárbara A. 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. Pressman Roger S., Mac Graw-Hill. Ingeniería de Software. Un enfoque práctico. 2001.

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

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

Glosario de Términos

CSS: Cascading Style Sheets u Hojas de Estilo en Cascada, es la tecnología desarrollada por el World Wide Web Consortium (W3C) con el fin de separar la estructura de la presentación.

Caso de prueba: es un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados.

Gestión de la configuración: conjunto de procesos destinados a asegurar la integridad de todos los productos obtenidos durante cualquiera de las etapas del desarrollo de un sistema de Información (SI), a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.

Interfaz: es el conjunto de comandos y/o métodos que permiten la intercomunicación del programa con cualquier otro programa o entre partes (módulos) del propio programa o elemento interno o externo.

Metodología: definición del proceso de investigación que sigue a la iniciación y permite sistematizar los métodos, procedimientos y las técnicas necesarias para llevarlas a cabo.

PostgreSQL: es un servidor de base de datos relacional orientada a objetos de software libre, liberado bajo la licencia BSD.

Peso Normalizado: resultado que se obtiene después de formular y aplicar reglas con el propósito de realizar en orden una actividad específica y aproximación ordenada de la misma para un beneficio y con la cooperación de todos los involucrados.

Requisito: Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.

SOA: la arquitectura orientada a servicios (SOA), es un marco de trabajo conceptual que permite a las organizaciones unir los objetivos del negocio con la infraestructura de las Tecnologías de Información (TI), integrando los datos y la lógica de negocio de sus sistemas separados.

Trazabilidad: conjunto de medidas, acciones y procedimientos que permiten registrar e identificar cada producto desde su origen hasta su destino final.

Validación: confirmación mediante examen y aportación de pruebas objetivas de que se cumplen los requisitos concretos para un uso determinado.

Anexos

Anexo 1: Entrevista a líderes de proyectos y especialistas en la actividad de GR en proyectos de la Universidad de las Ciencias Informáticas.

Tema: Conocimiento sobre la actividad de GR.

Nombre del Entrevistado: _____

Rol: _____

1. ¿En su proyecto realizan la Gestión de Requisitos?

Sí ___ No___

a) Seleccione por qué no la realiza:

- Desconocen que existe
- No lo consideran importante
- No existe la herramienta indicada
- Otras causas. Menciónelas: _____

2. ¿Ha tenido su proyecto alguna dificultad por una inadecuada Gestión de Requisitos?

Sí ___ No___

Mencione la(s) dificultad(es) _____

3. ¿En la actualidad, mediante qué forma su proyecto realiza la Gestión de Requisitos?

Manual___ Mediante una herramienta informática ____

a) La herramienta informática realiza el cálculo de la complejidad de los Requisitos Funcionales:

Sí ___ No___

Menciónelas_____

4. Conoce usted alguna(s) de la(s) herramienta(s) para la Gestión de Requisitos.

Sí ___ No ___

Menciónelas_____

Anexo 2: Carta de Aceptación del Cliente

UCI Informáticas

Acta de aceptación

Grupo de Investigación y Desarrollo de la herramienta para SARE

11 de abril de 2013

En cumplimiento de la Herramienta para determinar la complejidad de los requisitos funcionales de software para el Sistema de Administración de Requisitos (SARE) en función de la ejecución del Grupo de Investigación y Desarrollo de la herramienta para el sistema, se hace entrega del producto que se relaciona a continuación.

- Artefactos generados en el cumplimiento de la Solución (Modelo de Dominio, Especificaciones de Requisitos Funcionales, Diagrama de Clases del Diseño y Diseño de Casos de Prueba)
- Herramienta para determinar la complejidad de los requisitos funcionales de software.
- Informe del Trabajo de Diploma.

Entrega: Herramienta para determinar la complejidad de los requisitos funcionales de software.	**Recibe:** Asesor de la Subdirección de Desarrollo Calisoft.	
Nombre y Apellidos: Yoandi Díaz Ramos	**Nombre y Apellidos:** Enrique Pérez Rodríguez	**Cargo:** Jefe del Grupo de Investigación y Desarrollo de la herramienta para SARE en Calisoft.
Nombre y apellidos de los tutores: Ing. Daimara Mustelier Sanchidrian Ing. Tamara Rodríguez Sánchez	**Certifican:** MSc. Karina Perez Teruel **(analista principal)** Ing. Enrique Pérez Rodríguez **(arquitecto)**	
Declaro que la solución entregada cumple con el marco tecnológico definido y al margen de la Declaración de Autoría es cedida al Grupo de Investigación y Desarrollo de la herramienta para SARE en Calisoft. **Firma del autor principal:**	*Declaro que la solución entregada cumple con el marco tecnológico definido y ha sido transferida al Grupo Investigación y Desarrollo de la herramienta para SARE en Calisoft en beneficio del sistema.* **Firma del jefe proyecto:**	
Comentarios:		

1