

Universidad de las Ciencias Informáticas

Facultad 3



Herramienta para la determinación de la complejidad de los requisitos funcionales de software utilizando técnicas de soft computing.

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

Autor: Alejandro Alberto Ramírez Toranzo

Tutor(es): Ing. Tamara Rodríguez Sánchez

Ing. Yixander Yero Tarancón

Dra.C. Lizandra Arza Pérez

La Habana, junio de 2015

Agradecimientos

A mis padres que en todo momento importante de mi vida han estado presentes, dándome su apoyo incondicional. Gracias por existir y ser el pilar que me sostiene cuando estoy a punto de caer.

A mi futura esposa, la mejor persona que pude conocer para construir mi familia. Te agradezco por todo, me es imposible escribir lo que he llegado a ser gracias a ti. Gracias por darme el mejor regalo de todos, nuestro hijo. Gracias por ser como eres y por ser mi otra mitad, sin ti no existiría.

A mis hermanos Alberto y Luisito, a mi tía Trelly que ha sido como una madre, a mi tía Tere que me recibe como un rey siempre que la visito, a mi abuelo Mario, a mi tío Rodolfo, a mis primos Katy y Gaby. Gracias por su preocupación y por darme tantos momentos felices en familia.

A mi otra familia, en especial a mi suegra Rosa Elena que me recibió como si fuera su hijo. Gracias por tu constante preocupación y por ser alguien tan especial.

A mi súper tutora Tamara que ha sido increíble durante el desarrollo de mi tesis. Gracias por la gran ayuda que me has brindado en estos meses, gracias por todo.

A los grandes amigos que hice durante estos cinco años, en especial a Negrín que ha sido como un hermano para mí.

A todos mis compañeros de grupo que luchamos juntos para las parciales y finales, Negrín, Roli, Oda, Ara, Ana, Alicia, Jose, Bauta, Aniel, los jimaquas, Mariehys, Sosa y a Yaicel que fue como un tutor más para mí. Gracias a todos.

A todos los profesores que han contribuido en mi formación profesional. Gracias por su perseverancia y confianza.

Dedicatoria

A mi abuela Ángela, donde quiera que estés, gracias por tu amor y apoyo incondicional.

A mis padres, a Dai y a mi hijo Lucas, ustedes son la razón que me lleva a sonreírle a la vida cada día.

Declaración jurada de Autoría

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

Para que así conste firmo la presente a los ___ días del mes de ___ del año 2015.

Autor:

Alejandro Alberto Ramírez Toranzo

Tutores:

Ing. Tamara Rodríguez Sánchez

Ing. Yixander Yero Tarancón

Dra.C. Lizandra Arza Pérez

Datos de Contacto

Tutora:

- **Nombre y apellidos:** Ing. Tamara Rodríguez Sánchez.
- **Correo electrónico:** trodriguez@uci.cu
- **Situación laboral:** Especialista.
- **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 de Cedrux, Analista Principal del centro CEIGE, Facultad 3

Tutor:

- **Nombre y apellidos:** Ing. Yixander Yero Tarancón.
- **Correo electrónico:** yero@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:** Especialista de la Dirección de Calidad de Software, Jefe del grupo de Procesos de la Dirección de Calidad de Software.

Tutora:

- **Nombre y apellidos:** Dra.C. Lizandra Arza Pérez.
- **Correo electrónico:** lizandra@uci.cu
- **Situación laboral:** Directora de Investigación.
- **Institución:** Universidad de las Ciencias Informáticas (UCI).
- **Dirección:** Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros.

RESUMEN

Los requisitos son razones fundamentales tanto de éxito como de fracaso en el desarrollo de un software, es por esto que todo esfuerzo que permita reducir errores en los mismos resulta una acertada decisión. La evaluación por complejidad de los requisitos funcionales es presentada de forma cualitativa, dificultando una evaluación precisa de la misma. La presente investigación tiene como objetivo principal desarrollar una herramienta que tiene como base el Método para determinar la complejidad de los requisitos funcionales de software, en el cual se trata la incertidumbre de la información a través de la aplicación de técnicas de soft computing. Con la utilización de dicha herramienta, se logra una unificación de los criterios de múltiples expertos, teniendo en cuenta el juicio humano. Para corroborar la validez de la herramienta se aplicaron pruebas de caja blanca (camino básico), caja negra (partición de equivalencias) y pruebas de aceptación, así como la técnica de ladov para conocer el grado de satisfacción de los usuarios finales.

Palabras claves: complejidad, expertos, incertidumbre, requisitos funcionales de software.

ÍNDICE

Introducción	11
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.....	15
1.1 Ingeniería de Requisitos.....	15
1.2 Técnicas de Soft Computing	18
1.3 Herramientas de gestión de requisitos	21
1.4 Metodología de desarrollo, lenguaje de modelado y herramientas CASE	23
1.5 Frameworks, tecnologías y herramientas de desarrollo.....	27
1.6 Patrón de arquitectura.....	32
1.7 Patrones de diseño	33
1.8 Pruebas de software	34
1.9 Conclusiones parciales	36
CAPÍTULO 2: REQUISITOS, ANÁLISIS Y DISEÑO	37
2.1 Propuesta de solución.....	37
2.2 Modelo conceptual	37
2.3 Requisitos	38
2.4 Clases del diseño.....	44
2.1 Diagramas de secuencia.....	48
2.2 Modelo de datos.....	49
2.3 Patrones de diseño utilizados.....	50
2.4 Conclusiones parciales	52
CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN	54
3.1 Modelo de implementación.....	54
3.2 Código fuente.....	55
3.3 Diagrama de despliegue	57
3.4 Validación del diseño propuesto.....	57
3.5 Pruebas de software aplicadas a la herramienta	64
3.6 Validación de satisfacción del usuario. Aplicación de la Técnica de ladov.	70
3.7 Conclusiones parciales	73
Conclusiones generales	74

Recomendaciones.....	75
Referencias Bibliográficas.....	76
Anexos.....	79
Anexo 1: Metodología AUP-UCI	79
Anexo 2: GRASP	80
Anexo 3: Diagrama de clases del diseño	81
Anexo 4: Acta de aceptación	82
Anexo 5: Encuesta.....	83

Índice de Figuras y Tablas

Figura 1 Fases e iteraciones. [21].....	24
Figura 2 Estructura del patrón MVC. [40]	33
Figura 3 Modelo conceptual (elaboración propia).....	37
Figura 4 Paquete Vista (elaboración propia).....	45
Figura 5 Paquete Controlador (elaboración propia).....	46
Figura 6 Paquete Modelo (elaboración propia).	47
Figura 7 Diagrama de secuencia del Gestionar Requisitos Escenario: Importar Requisito (elaboración propia).	48
Figura 8 Diagrama de secuencia del Gestionar Complejidad Escenario: Calcular complejidad (elaboración propia).	49
Figura 9 Modelo de datos (elaboración propia).	49
Figura 10 Diagrama de componentes (elaboración propia).	54
Figura 11 Ejemplo de comentarios en el código (elaboración propia).....	56
Figura 12 Diagrama de despliegue (elaboración propia).....	57
Figura 13 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos (elaboración propia).....	59
Figura 14 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad (elaboración propia).	60
Figura 15 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación (elaboración propia).....	60
Figura 16 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización (elaboración propia).....	60
Figura 17 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos (elaboración propia).	62
Figura 18 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento (elaboración propia).	62
Figura 19 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento (elaboración propia).....	63
Figura 20 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de pruebas (elaboración propia).....	63
Figura 21 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización (elaboración propia).....	63

Figura 22 Código: calcularComplejidad (elaboración propia).	65
Figura 23 Grafo de flujo asociado al algoritmo calcularComplejidad (Requisito \$requisito) (elaboración propia).	65
Figura 24 Resultados de las pruebas de caja negra (elaboración propia).....	69
Tabla 1 Herramienta/Indicadores (elaboración propia).	23
Tabla 2 Complejidad de RF (elaboración propia).....	39
Tabla 3 Descripción del diseño de clases paquete Vista (elaboración propia).	45
Tabla 4 Descripción del diseño de clases paquete Controlador (elaboración propia).	46
Tabla 5 Descripción del diseño de clases paquete Modelo (elaboración propia).	48
Tabla 6 Diccionario de datos tabla Requisito (elaboración propia).	50
Tabla 7 Diccionario de datos tabla Experto (elaboración propia).	50
Tabla 8 Tamaño operacional de clases TOC (elaboración propia).	58
Tabla 9 Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC (elaboración propia).	59
Tabla 10 Relaciones entre clases RC (elaboración propia).	61
Tabla 11 Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica RC (elaboración propia).	62
Tabla 12 Caminos básicos del flujo (elaboración propia).	66
Tabla 13 Caso de prueba para el requisito Importar requisitos (elaboración propia).....	69
Tabla 14 Resultados Evaluación de requisitos.xls (elaboración propia).	70
Tabla 15 Resultados COMREQ (elaboración propia).....	70
Tabla 16 Cuadro Lógico de ladov. (Modificado por el autor).	71
Tabla 17 Resultados de la aplicación de la técnica de ladov (elaboración propia).	72

Introducción

La industria de software continúa su auge y desarrollo en la sociedad, sin embargo, los resultados alcanzados no cubren las expectativas inicialmente deseadas debido básicamente a que, la cantidad de recursos a consumir (en tiempo principalmente) es alta y el trabajo realizado casi nunca tiene la calidad requerida. Los proyectos se concluyen en fecha posterior a lo planificado y los problemas no se detectan a tiempo.

La Ingeniería de Requisitos (IR) es una de las principales etapas en el proceso de desarrollo de software, esta permite o no una adecuada gestión de los requisitos, actividad en la que se plantea el “qué” y el “cómo” de lo que se va a hacer. Una buena aplicación de la IR resultará en un software con mayor calidad y en una reducción de costos, recursos y atrasos del proyecto que lo desarrolle. El informe CHAOS, elaborado por The Standish Group¹, es el informe más famoso sobre el éxito y fracaso de los proyectos desarrolladores de software. Según el CHAOS Report 2013 [1], la cifra de proyectos exitosos ha aumentado sólo del 35% al 39%, desde el anterior informe realizado en 2012. Se resume que los factores que inciden en los proyectos exitosos son la participación del usuario, clara exposición de los requisitos y una adecuada planificación; no siendo de esta forma en el caso de los proyectos en riesgo y los fracasados, donde no se incluyen todos los requisitos planificados y donde los requisitos son incompletos, respectivamente. Esto evidencia que el tratamiento que se le dé a los requisitos de software, incidirá en el éxito o fracaso de los proyectos, es por ello que la IR juega un papel fundamental en el proceso de desarrollo de software.

Dentro de la IR una de las actividades a realizar es la evaluación de los requisitos, lo cual permite determinar la complejidad de los RF, partiendo de la especificación correcta de cada uno de estos [2]. Esta información resultante es brindada al cliente en términos cualitativos, sin embargo dado que es información cualitativa no se tiene en cuenta la incertidumbre contenida en el criterio de varios expertos. En la actualidad existen varios sistemas que gestionan RF de software, sin embargo, carecen de una funcionalidad en específico capaz de determinar la complejidad de los mismos. En Cuba las empresas desarrolladoras de software no están al margen de la situación que se presenta en los estudios mundiales. En aras de aunar los esfuerzos individuales que han venido realizando diversas instituciones del país para alcanzar una fortaleza que permita incursionar con más efectividad en los mercados extranjeros, fue creado Incusoft (Industria Cubana del Software). Un ejemplo ilustrativo del desarrollo de software en Cuba lo constituye la Universidad de las Ciencias Informáticas (UCI).

La UCI enfoca su desarrollo de software a lograr una mejor aplicación de los procesos que como parte de la actividad productiva se llevan a cabo en la misma, dentro de estos un área de suma importancia es la Administración de requisitos. En pos de cumplir lo antes expuesto, a partir del

¹ The Standish Group es una organización asesora de investigación primaria que se centra en el rendimiento de proyectos de software.

año 2008 se lleva a cabo el Programa de Mejoras CMMI² nivel 2 para desarrollo y en el año 2010 surge el método “Evaluación por complejidad de los RF de software” como parte de la mejora de los procesos relacionados con la actividad productiva. Con este método se realiza una valoración cualitativa (alta, media y baja) de la complejidad basándose en:

- Variables que definen la complejidad de un requisito.
- Métodos matemáticos que ofrecen una puntuación para cada requisito.
- Hoja de Cálculo del paquete Microsoft Office donde se realizan los cálculos correspondientes a cada requisito.

El resultado obtenido incide en las estimaciones que definen la planificación de los proyectos, es decir, en las relacionadas con el tiempo, esfuerzo, recursos y costos de los mismos.

Sin embargo, como parte de la mejora de procesos que se realiza en la UCI, en específico, de la utilización y ejecución de este método, se identificaron algunas deficiencias que impiden la realización de un óptimo análisis de la complejidad de los RF de software [3]:

- No tiene en cuenta que es un problema en el que pueden intervenir múltiples expertos.
- Las variables no están directamente relacionadas con un RF dado que en su mayoría corresponden a un requisito no funcional.
- No tiene en cuenta la incertidumbre como parte de la información cualitativa que se emite al dar las clasificaciones de la complejidad.
- En la herramienta no todas las celdas presentan la formulación llevando al usuario a realizar los cálculos fuera de la misma o a invertir tiempo en arreglar este problema manualmente en la hoja de cálculo.
- En la herramienta si se deja una celda en blanco realiza una división por cero que no está definida, originando estimaciones erróneas.
- La herramienta no permite la generación de reportes lo cual afecta la toma de decisiones en los proyectos.

Teniendo en cuenta lo antes planteado, en la UCI se llevó a cabo como parte del programa de mejoras una investigación para definir un método que disminuyera los errores del método actual [3] haciéndose necesario la implementación de una herramienta que automatice el nuevo método y de solución a los problemas antes mencionados.

De lo anteriormente planteado se define como **problema a resolver**:

² Capability Maturity Model Integration: Integración de modelos de madurez de capacidades.

¿Cómo mejorar el proceso de determinar la complejidad de los RF de software en los proyectos de la UCI de manera que se minimicen las deficiencias que presenta la herramienta actual?

El **objeto de estudio** del presente trabajo lo constituyen las herramientas para la gestión de requisitos, centrando el **campo de acción** en las herramientas basadas en técnicas de soft computing para determinar la complejidad de los RF de software.

Para mitigar las deficiencias de la herramienta actual se traza como **objetivo general**: Desarrollar una herramienta basada en técnicas de soft computing que mejore el proceso de determinar la complejidad de los RF de software en los proyectos de la UCI de manera que se minimicen las deficiencias que presenta la herramienta actual.

Como **idea a defender** se plantea que si se desarrolla una herramienta basada en técnicas de soft computing entonces mejorará el proceso de determinar la complejidad de los RF de software en los proyectos de la UCI y se minimizarán las deficiencias que presenta la herramienta actual.

Para darle cumplimiento al objetivo general se desarrollarán los siguientes **objetivos específicos**:

- OE1. Realizar un estudio del estado del arte para la elaboración del marco teórico alrededor del objeto de estudio.
- OE2. Desarrollar la IR para la identificación de las funcionalidades y características de la herramienta.
- OE3. Diseñar la estructura de componentes, clases y datos aplicando los patrones de diseño, algoritmos y técnicas consideradas como necesarias en el estudio.
- OE4. Implementar la herramienta siguiendo las técnicas de programación estudiadas en la plataforma de desarrollo seleccionada.
- OE5. Validar la solución propuesta.

Como tareas generales de la investigación se tienen:

- Estudio de las diferentes herramientas de gestión de requisitos.
- Análisis de las técnicas de soft computing para el tratamiento de la incertidumbre.
- Diseño de las clases y componentes aplicando patrones de arquitectura y de diseño.
- Implementación de una herramienta que permita la determinación de la complejidad de los RF de software.
- Evaluación del resultado de la herramienta a través de la aplicación de pruebas y proposición de mejoras a la misma.

Como **posible resultado** se espera una herramienta basada en técnicas de soft computing para determinar la complejidad de los RF de software en los proyectos de la UCI.

Para la realización de la presente investigación se aplicaron los siguientes métodos:

Métodos teóricos:

Analítico-Sintético: El análisis y la síntesis no existen independientemente uno del otro. En realidad el análisis se produce mediante la síntesis y a su vez, la síntesis se produce sobre la base de los resultados dados previamente por el análisis [4], esto permitió analizar documentos, teorías, permitiendo la obtención y descomposición de los elementos más importantes que se relacionan con la complejidad de los RF.

Histórico-Lógico: El método histórico estudia la trayectoria real de los fenómenos y acontecimientos en el decursar de su historia para comprobar teóricamente cómo se comporta la evaluación por complejidad de los RF teniendo en cuenta que la información resultante no es precisa. [4]

Inductivo-Deductivo: Para generalizar la teoría estudiada y llegar a conclusiones específicas sobre la aplicación de las técnicas de soft computing en el proceso de determinación de la complejidad de los RF.

Modelación: Es una reproducción simplificada de la realidad que cumple una función heurística, ya que permite descubrir y estudiar nuevas relaciones y cualidades del objeto de estudio [4] para elaborar los artefactos establecidos en la metodología de desarrollo durante la disciplina de análisis y diseño.

Métodos empíricos:

Observación: Se logra una percepción general sobre el objeto de estudio, específicamente el comportamiento de la herramienta definida actualmente en la UCI, resultando en un diagnóstico sobre la usabilidad de la misma.

Entrevistas: A especialistas que utilizan la herramienta en la UCI para justificar las deficiencias que presenta.

Medición: Comparar resultados medibles a través de la utilización de la herramienta resultado de la presente investigación en un proyecto real.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se realiza un estudio actual referente a la IR, incluyendo sus principales conceptos. Se describe la incertidumbre como elemento fundamental a tener en cuenta dado la información cualitativa que se maneja al evaluar los RF en cuanto a su complejidad para el desarrollo de un proyecto de software. Por último se ofrece un análisis sobre las herramientas de gestión de requisitos en Cuba y a nivel mundial, Metodología, Lenguajes de Programación, Entornos de Desarrollo Integrado (IDE), Gestores de Base de Datos, Framework y las Pruebas de Software, lo cual permitirá arribar a conclusiones que apoyen el desarrollo de la herramienta de la presente investigación.

1.1 Ingeniería de Requisitos

Toda actividad relacionada con el desarrollo de software conlleva a la aplicación de la IR. Esta constituye la disciplina principal para guiar el proceso de captura, documentación y mantenimiento de los requisitos.

La IR es una de las disciplinas más importantes en el proceso de desarrollo de software. 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. Varias han sido las definiciones referentes a esta disciplina, coincidiendo de alguna manera con los siguientes elementos [5], [6], [7]:

- Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software. [7]
- Proceso de comunicación entre los clientes y usuarios del software y los desarrolladores del mismo. [5]
- Es la disciplina para desarrollar una especificación completa, consistente y no ambigua, la cual servirá como base para acuerdos comunes entre todas las partes involucradas y en donde se describen las funciones que realizará el sistema. [6]

El autor coincide con cada uno de los elementos detallados anteriormente. Existe un elemento esencial en esta disciplina y es el relacionado con los RF, sin estos no existiría un producto de software. La clasificación de estos dentro del proceso de desarrollo de software permite un mejor entendimiento e implementación de los mismos.

Clasificación de los requisitos

Los requisitos de software generalmente pueden clasificarse en dos grupos: los funcionales y los no funcionales. Se estudiaron varias definiciones de requisito, tales como:

- Los requisitos son una especificación de lo que debe ser implementado. Son descripciones de cómo el sistema debe comportarse. Ellos pueden ser una limitación en el proceso de desarrollo del sistema. [8]
- Capacidad, característica o factor de calidad de un sistema mediante el cual se pretende cumplir con determinadas necesidades o restricciones operativas, aportando un valor y una utilidad para un cliente o usuario dentro del marco de solución de un problema en un entorno real. [9]
- Una condición o capacidad que una aplicación debe cumplir o tener para resolver un problema o alcanzar un objetivo. [10]

El autor asume la definición de la IEEE que enmarca a un requisito como una condición que una aplicación debe cumplir o tener para resolver un problema o alcanzar un objetivo, resumiendo en gran medida lo planteado por los diferentes autores citados.

En la presente investigación se tienen en cuenta los RF. Definiéndose los mismos como:

- Forma en la que el sistema a implementar debe comportarse ante situaciones particulares y cómo reaccionar ante entradas particulares, es decir, describe lo que el sistema debe hacer. En algunos casos especifican lo que el sistema no debe hacer. [5]
- Especifica una función que un sistema o componente del sistema debe ser capaz de realizar. [10]
- Definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema será capaz de realizar así como las transformaciones que el sistema realiza sobre las entradas para producir salidas. [11]

El autor coincide con todas las definiciones presentadas, definiendo como RF la capacidad que posee un sistema de adaptarse a cualquier entrada o situación específica para de esta forma generar salidas, describiendo el qué y no el cómo realizar estas transformaciones.

Durante la gestión de los RF se tienen en cuenta dos clasificaciones de suma importancia y son las relacionadas con la prioridad y la complejidad de los mismos. La prioridad está relacionada con el orden de implementación y la complejidad apoya los elementos relacionados con el tiempo, esfuerzo, recursos y costo de un proyecto de desarrollo de software. Ambas clasificaciones influyen en la planificación de un proyecto de desarrollo de software.

En la presente investigación se tendrá en cuenta la clasificación por complejidad de los RF asumiendo la definición según Pressman: “el grado en que un sistema o componente tiene un diseño o aplicación que es difícil de entender y verificar”. [2]

Al referirse a la complejidad de un requisito, se está haciendo alusión a la característica que describe la dificultad de diseño e implementación de este dentro del proceso de desarrollo de software. Es de gran importancia para los expertos clasificar los RF de software según su complejidad, dado que permite brindar un mejor tratamiento de los mismos durante el desarrollo de cualquier producto, sirviendo de soporte para la toma de decisiones de las fases posteriores.

Técnicas de captura y validación de los requisitos

Existen varias técnicas de captura de requisitos, para las cuales se debe tomar en cuenta las características propias del proyecto en particular que se esté desarrollando. Esto permitirá obtener de forma clara las necesidades planteadas por los clientes y así lograr la satisfacción de estos.

Algunas de las técnicas a utilizar para la captura de requisitos son [12]:

- Entrevistas: Es una de las técnicas más usadas en la captura de requisitos. Consiste en establecer una conversación entre personas de ambas partes (cliente y equipo de desarrollo). Permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural. [13]
- 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. [13]
- Cuestionarios: Esta técnica requiere que el analista conozca el ámbito del problema en el que está trabajando. Consiste en redactar un documento con preguntas cuyas respuestas sean cortas y concretas, o incluso cerradas por unas cuantas opciones en el propio cuestionario. [13]
- Casos de uso: Permiten mostrar el contorno (actores) y el alcance (RF expresados como casos de uso) de un sistema. Un caso de uso describe la secuencia de interacciones que se producen entre el sistema y los actores del mismo para realizar una determinada función. [13]

Para la captura de los requisitos que guiarán el desarrollo de la herramienta en la presente investigación, se van a aplicar las técnicas entrevistas y tormenta de ideas.

Las técnicas de validación de los requisitos constituyen un elemento fundamental dado que permiten lograr un entendimiento común entre todos los involucrados.

Algunas de las técnicas a emplear consisten en: [14]

- 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. [14]
- Auditorías: Consisten en un chequeo de los resultados contra una lista de chequeo predefinida o definida a comienzos del proceso. [14]
- Matrices de trazabilidad: Esta técnica consiste en marcar los objetivos del sistema y chequearlos contra los requisitos del mismo. Es necesario reconocer qué objetivos cubre cada requisito, de esta forma se podrán detectar inconsistencias u objetivos no cubiertos. [14]

- Prototipos de interfaz de usuario: Ayudan a identificar, comunicar y probar una herramienta antes de crearla. La realización de los mismos, logra un entendimiento común entre el cliente y los desarrolladores. [14]

El autor hará uso de la técnica de validación relacionada con los prototipos de interfaz de usuario, lo cual va a permitir una mejor comunicación y entendimiento por parte de los involucrados.

La complejidad de los RF es una información cualitativa, por lo que no puede ser evaluada de forma precisa. Por ello se hace necesaria una forma de tratar la incertidumbre albergada en dicha información.

1.2 Técnicas de Soft Computing

Soft computing, brinda un marco adecuado para una representación más cercana a la realidad del problema, que pueda representar el razonamiento humano y permita incorporar sus valoraciones con tratamiento de la incertidumbre. Se puede considerar soft computing como un enfoque multidisciplinario donde la teoría de conjuntos borrosos facilita la representación del razonamiento humano y su capacidad de aprender en un ambiente de incertidumbre e imprecisión [15]. Las técnicas de soft computing tienen su base en los conceptos asociados a la teoría de conjuntos borrosos, la lógica borrosa y el modelado lingüístico de la información.

La presente investigación se basa en el Método para determinar la complejidad de los RF de software [3], para el desarrollo de la herramienta se hará uso de los principales conceptos y definiciones referentes a las técnicas de soft computing aplicadas en este método [3]. A continuación se exponen los elementos fundamentales a tener en cuenta para la implementación de la herramienta.

- **Variables que definen la complejidad de los RF de software.** [3]
 - **Interfaces:** Se aplica a requisitos que presenten algún tipo de complejidad en su interacción con los siguientes elementos, considerados subcomponentes.
Humanas (formularios, informes) Equipo (Tomógrafos, Rayos X)
Programación (Programas externos necesarios para apoyar el producto)
Comunicación (Protocolos de comunicación que serán utilizados)
Atributos (Se refiere a la complejidad que puede agregarle la cantidad de atributos contenidos en una interfaz)
 - **Facilidad de cambio:** Esfuerzo específico de diseño e implementación del requisito para facilitar cambios futuros durante el desarrollo del producto.
 - **Dependencia con otros requisitos:** Describe el grado de relación de un requisito con otros, según la cantidad de requisitos con los que guarde algún tipo de dependencia para su implementación.
 - **Requisitos asociados:** Se refiere a la descomposición en sub-funciones de un requisito más general y que no guardan dependencia entre sí.

- **Transacciones:** Es la interacción con una estructura de datos compleja, compuesta por varios procesos que se han de aplicar uno después del otro.
- **Diferentes comportamientos:** Se refiere al comportamiento que puede tener el requisito ante determinadas situaciones, en dependencia de ello se recogerá más información.
- **Restricciones:** Se refiere a los requisitos asociados a las restricciones de diseño, implementación, interfaces, físicas y reglas de negocio.
 - **Restricciones de diseño:** Limitar el diseño y declarar los requisitos sobre el enfoque que debe tenerse en cuenta en el desarrollo del sistema.
 - **Restricciones de implementación:** Poner límites al proceso de generación de código o de construcción (estándares requeridos, lenguajes, herramientas o plataforma)
 - **Restricciones de interfaz:** Son requisitos para interactuar con sistemas externos, describiendo los protocolos o la naturaleza de la información que debe ser transferida a través de la interfaz.
 - **Restricciones físicas:** Afectan el hardware o el empaquetado del sistema (forma, tamaño y peso)
 - **Reglas del negocio:** Son las políticas, normas, estatutos, acuerdos, resoluciones o cualquier tipo de decisión que gobierna la forma en que la institución opera. Ellas restringirán los pasos descritos en el flujo del caso de uso.
- **Variables lingüísticas y conjunto de términos lingüísticos. [3]**
 - La influencia de las variables sobre los RF estará representada por la variable lingüística "*Influencia de la variable*" y el conjunto de términos lingüísticos: No influye, Muy baja, Baja, Media, Alta. Por lo tanto la Influencia de las variables se delimita por un término lingüístico del conjunto $S = \{s_0 = \text{No influye}, s_1 = \text{Muy Baja}, s_2 = \text{Baja}, s_3 = \text{Media}, s_4 = \text{Alta}\}$.
 - La complejidad de los RF de software estará dada por la variable lingüística "*Complejidad del requisito funcional*" y el conjunto de términos lingüísticos: Muy baja, Baja, Media, Alta, Muy Alta. Por lo tanto la Complejidad de los requisitos se delimita por un término lingüístico del conjunto $S = \{s_0 = \text{Muy baja}, s_1 = \text{Baja}, s_2 = \text{Media}, s_3 = \text{Alta}, s_4 = \text{Muy Alta}\}$.

Descripción de los componentes del método por los cuales se guiará la herramienta a desarrollar. [3]

La valoración de las variables que definen la complejidad de un requisito funcional se realiza con el objetivo de llevar a cada una de las variables a una misma variable lingüística "*Influencia de la variable*".

La valoración cuenta de tres pasos fundamentales:

1. Criterio de los expertos, basado en las entradas del método propuesto, así como en su conocimiento y experiencia en el área de Administración de requisitos.
2. Asociar a partir del criterio de los expertos las variables al término lingüístico de la variable lingüística “*Influencia de la variable*”.

$$S = \{s_0 = \text{No influye}, s_1 = \text{Muy baja}, s_2 = \text{Baja}, s_3 = \text{Media}, s_4 = \text{Alta}\}$$

3. Valorar a partir del criterio de los expertos los pesos asociados para cada una de las variables.

El peso estará dado en una escala del 1 al 5, el mismo radicaré en la importancia que le atribuye el experto a la presencia de la variable en el requisito funcional.

La **determinación de la complejidad** se realiza a partir de haber relacionado para cada variable los términos lingüísticos y los pesos asignados por los expertos, mediante el operador de agregación de media ponderada adaptado a 2-tuplas. [3]

- **Cálculo de influencia de las variables** [3]: para cada variable se operará teniendo en cuenta los siguientes elementos:
 1. Un valor $\beta \in [0, g]$ partiendo de que el conjunto de términos lingüísticos está representado por $S = \{s_0, \dots, s_g\}$, utilizando para el método propuesto un total de cinco términos $S = \{s_0, s_1, s_2, s_3, s_4\}$, por lo que el valor $\beta \in [0, 4]$.
 2. Pesos asociados w_i para cada variable, los cuales se asocian a los términos lingüísticos seleccionados en dependencia del criterio de los expertos.

Teniendo cada uno de estos elementos el cálculo de influencia de cada variable se realizaría mediante:

$$I_v = \sum_{i=1}^n \beta_i \cdot w_i \quad (1)$$

Donde I_v representa la influencia de la variable que se esté analizando, β_i es el valor del término lingüístico y w_i es el peso asignado por cada experto, donde n representa la cantidad de expertos.

- **Determinar complejidad del Requisito funcional** [3]: es un paso clave para lograr la salida del método propuesto en la investigación, para ello se tendrá en cuenta la influencia de cada variable.

Para determinar la complejidad de modo que los cálculos se realicen en una sola operación se utilizará el operador de media ponderada adaptado para 2-tuplas.

$$C_{rf} = \Delta \left(\frac{\sum_{i=1}^m \Delta^{-1}(r_i, a_i) \cdot w_i}{\sum_{i=1}^m w_i} \right) = \Delta \left(\frac{\sum_{i=1}^m I_{v_i}}{\sum_{i=1}^m w_i} \right) \quad (2)$$

Donde C_{rf} es el valor de la complejidad del requisito funcional a partir del análisis de cada una de las variables, $\sum_{i=1}^m I_{v_i}$ el total de la suma de la influencia de cada una de las variables, $\sum_{i=1}^m w_i$

representa el peso total asignado a cada variable, m representa la cantidad de variables.

Transformar resultados en términos lingüísticos [3]: La complejidad es una información cualitativa que se clasificará teniendo en cuenta el conjunto de términos lingüísticos $S = \{s_0 = \text{Muy baja}, s_1 = \text{Baja}, s_2 = \text{Media}, s_3 = \text{Alta}, s_4 = \text{Muy Alta}\}$.

Para transformar los resultados en términos lingüísticos se hará uso de la representación en 2-tuplas (s_i, α) , donde: s_i representa el término lingüístico, y α_i es un número que expresa el valor de la distancia desde el resultado original al índice del término lingüístico s_i más cercana en el conjunto de términos lingüísticos S , es decir, su traslación simbólica. [3]

Con la aplicación de las técnicas de soft computing para el desarrollo de la herramienta que se propone, se logra la unificación del criterio de diferentes expertos. Esto permite el entendimiento de todos los involucrados al obtener una única salida que refleje el razonamiento de los mismos. Dado que es un problema donde interviene el juicio humano en todo momento, se trata la incertidumbre contenida en los criterios realizados sobre los requisitos a evaluar.

1.3 Herramientas de gestión de requisitos

Con el objetivo de facilitar las tareas del desarrollo de software, surgen herramientas informáticas que agilizan la labor en la IR y sirven de apoyo para los desarrolladores, desde el principio hasta el final del proceso.

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 multiplataforma. Se hizo un estudio de algunas herramientas que gestionan requisitos en las cuales se evaluaron los indicadores siguientes:

1. Complejidad de los Requisitos.
2. Captura e identificación de requisitos.
3. Análisis de trazabilidad.
4. Gestión de la configuración.
5. Libre.
6. Técnicas de soft computing.

Internacionales:

- **Rational RequisitePro** (IBM) [16]
 - Promueve el poder de una base de datos y la libertad de la aplicación de Microsoft Word para una administración más efectiva de los requisitos.
 - Permite vincular requisitos relacionados, de modo que cuando ocurra un cambio en un requisito se podrá ver su impacto en otros requisitos relacionados.

- Permite organizar y priorizar los requisitos, así como hacer la trazabilidad entre estos.
- **IBM Rational DOORS (IBM) [17]**
 - Gestiona los requisitos de forma colaborativa, intuitiva y escalada.
 - Implementa la trazabilidad a través de requisitos, diseños y pruebas.
 - Administra cambios y analiza el impacto de los cambios en los requisitos relacionados, diseños y pruebas.
- **OSRMT (Open Source Requirements Management Tool) [18]**
 - Permite la descripción avanzada de diversos tipos de requisitos y garantiza la trazabilidad entre todos los documentos relacionados con la ingeniería de requisitos (funcionalidades, requisitos, casos de uso, casos de prueba).
 - Realiza la gestión de la configuración: versionado y registro de los cambios realizados en los diferentes elementos.
 - Realiza la gestión de usuarios y permisos.
- **GatherSpace [19]**
 - GatherSpace™ es una solución de gestión de requisitos en la nube que promueve la colaboración entre las empresas y de equipos técnicos en la gestión de los requisitos cambiantes a través del desarrollo de software y del ciclo de vida del producto.
 - Crea grupos de paquetes funcionales y asocia características de alto nivel.
- **Heler [20]**
 - Determina el tipo de requisito, especificación, importancia que tiene un requisito en términos de implementación a través de la asignación de prioridad, urgencia y estado.
 - Valida si los requisitos creados satisfacen los objetivos mediante la matriz de trazabilidad que permite describir y seguir la vida de un requisito.

Nacionales:

Herramienta en Excel que implementa un método para la evaluación por complejidad de los RF de software. Actualmente es la herramienta que integra el expediente de proyecto, definido por el Programa de Mejoras para la UCI, la misma calcula la complejidad de los requisitos a partir de un conjunto de variables preestablecidas para dar una evaluación final de forma cualitativa (Alta, Media, Baja).

A continuación se muestra un resumen del estudio realizado, ver Tabla 1:

Herramientas	1	2	3	4	5	6
Rational RequisitePro	-	X	X	X	-	-
IBM Rational DOORS	-	X	X	X	-	-
OSRMT	-	X	X	X	X	-
GatherSpace	-	X	-	X	-	-
Heler	-	X	X	-	X	-
Excel/UCI	X	X	-	-	-	-

Tabla 1 Herramienta/Indicadores (elaboración propia).

Una vez realizado el estudio de estas herramientas se pudo constatar que solo la implementada en el Excel realiza el cálculo de la complejidad de los RF de software, pero como se detalló en la problemática de la investigación no tiene en cuenta que es un problema en el que pueden intervenir múltiples expertos. Las variables no están directamente relacionadas con un RF dado que en su mayoría corresponden a un requisito no funcional. Al dar las clasificaciones de la complejidad no se tiene en cuenta la incertidumbre como parte de la información cualitativa. Algunas celdas no presentan la formulación llevando al usuario a realizar los cálculos fuera de la herramienta o a invertir tiempo en arreglar este problema manualmente en la hoja de cálculo. Si se deja una celda en blanco realiza una división por cero que no está definida, originando estimaciones erróneas.

Por lo que se concluye que ninguna de las herramientas antes analizadas hace uso de técnicas de soft computing para realizar el cálculo. De ahí la importancia de desarrollar una herramienta que permita la determinación de la complejidad de los RF.

1.4 Metodología de desarrollo, lenguaje de modelado y herramientas CASE

Metodología de desarrollo

Para la realización de la presente investigación, se hace uso de la Metodología de desarrollo para la actividad productiva en la UCI. La misma se considera una variación del Proceso Unificado Ágil (AUP). Se conforma por Fases, Disciplinas, Roles y Productos de trabajos. Ver Anexo 1.

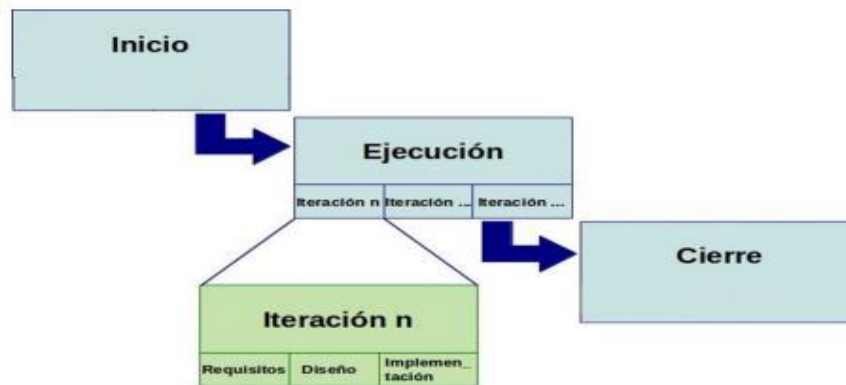


Figura 1 Fases e iteraciones. [21]

Tiene entre sus características que es Iterativa-Incremental y utiliza técnicas de modelado ágil. El autor enmarca la presente investigación en las fases Inicio y Ejecución, debido a que la fase de Cierre es solo aplicable a proyectos productivos. Las disciplinas que intervienen en la realización de la investigación son: Requisitos, Análisis y diseño, Implementación y Pruebas internas. Los productos de trabajos a generar son: modelo conceptual, modelo de diseño, descripción de requisitos, evaluación de requisitos, especificación de requisitos y diseño de casos de prueba.

Para modelar los sistemas, la metodología propone utilizar el Lenguaje unificado de modelado (UML).

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. Se ha convertido en el estándar de facto 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. [22]

Teniendo en cuenta el lenguaje de modelado a utilizar, a continuación se realiza un estudio sobre algunas herramientas CASE que lo aplican.

Herramientas 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 que reduce 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 de software.

Visual Paradigm 8.0 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 y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. [23]

Dentro de las principales características que posee esta herramienta están:

- La disponibilidad de integrarse en los principales Entornos de desarrollo integrado (IDE) como Netbeans.
- Soporta una gama de lenguajes en la Generación de Código e Ingeniería Inversa, fundamentalmente PHP.
- 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.

Rational Rose: Es una herramienta de producción y comercialización establecida por Rational Software Corporation (actualmente parte de IBM). Rose es un instrumento operativo conjunto que utiliza UML como medio para facilitar la captura de dominio de la semántica, la arquitectura y el diseño. No es gratuito, se debe hacer un previo pago para poder adquirir el producto. [24]

- Aplica los patrones de diseño.
- Admite la integración con otras herramientas de desarrollo (IDEs).

Para el modelado el autor hace uso de la herramienta Visual Paradigm 8.0 para UML dado que es libre y de fácil uso. Se destaca por su interoperabilidad con otras herramientas facilitando el trabajo para economizar en recursos asociados al mismo. Esta herramienta es la más utilizada en los proyectos productivos de la UCI lo cual permite que no sea necesario realizar un previo adiestramiento de la misma.

Una vez que se realiza el modelado del sistema, es muy importante seleccionar el lenguaje de programación idóneo para implementarlo.

Lenguajes 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. [25]

A continuación se realiza un estudio de los lenguajes PHP5 y Java.

PHP5

A medida que internet fue creciendo y sus funciones se ampliaron, las acciones requeridas también se complejizaron. Ya no alcanzaba con presentar el texto en una página y definir su estilo como proponía el HTML³. Fue así como surgieron lenguajes que permitían ampliar sus funciones, como es el caso de PHP. [26]

- Pertenece al software licenciado como GNU⁴, la licencia del sistema Linux; lo que permite su distribución gratuita y que la comunidad mejore el código. [26]
- Se trata de un lenguaje que se puede lanzar en casi todas las plataformas de trabajo (Windows, Linux, Mac,...). [26]
- Basado en C y en Perl, se ha diseñado pensando en darle la máxima versatilidad y facilidad de aprendizaje, por encima de la rigidez y coherencia semántica. [26]
- Dispone de un enorme número de extensiones que permiten ampliar las capacidades del lenguaje, facilitando la creación de aplicaciones web complejas. [26]

Java

La principal característica de Java es la de ser un lenguaje compilado e interpretado. Todo programa en Java ha de compilarse y el código que se genera (bytecodes) es interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma. [27]

- Lenguaje orientado a objetos de propósito general. [27]
- Se puede utilizar para construir cualquier tipo de proyecto. [27]
- Es un lenguaje multiplataforma y de amplio uso en software libre, actualmente bajo la licencia GPL⁵ de GNU. [27]

³ Siglas de HyperText Markup Language («lenguaje de marcas de hipertexto»), hace referencia al lenguaje de marcado para la elaboración de páginas.

⁴ GNU's Not Unix (GNU no es Unix).

El autor selecciona el lenguaje PHP5 para la implementación de la herramienta. Su selección se basa en la versatilidad de este lenguaje y en la facilidad para crear aplicaciones complejas, además posee una amplia gama de manuales y librerías que soportan lo antes expuesto. Es un lenguaje adaptable a cualquier entorno de desarrollo.

Para promover las buenas prácticas de desarrollo de software se hace necesario utilizar frameworks, tecnologías y herramientas. A continuación se realiza un estudio al respecto.

1.5 Frameworks, tecnologías y herramientas de desarrollo

Frameworks

El término framework se refiere a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que se le pueden añadir las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente. Un framework Web, por tanto, se puede definir como un conjunto de componentes (por ejemplo clases en java y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web. [28]

ExtJS 4.0.7: Es un framework JavaScript para la creación de aplicaciones enriquecidas del lado del cliente. Sus características principales son: [29]

- Gran desempeño, componentes de interfaz de usuario personalizables, con buen diseño y documentación.
- Es compatible con la licencia GNU GPL.

Symfony 2: Symfony es un framework PHP construido con varios componentes independientes creados por el proyecto Symfony. [30]

- Su código y el de todos los componentes y librerías que incluye, se publican bajo la licencia MIT⁶ de software libre. La documentación del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos.
- Integra plenamente uno de los frameworks ORM⁷ más importantes dentro de los existentes para PHP llamado Doctrine, el cual es el encargado de la comunicación con la base de datos, permitiendo un control casi total de los datos sin importar el gestor.

⁵ General Public License (Licencia Pública General).

⁶ Es una de tantas licencias de software que ha empleado el Instituto Tecnológico de Massachusetts (MIT, Massachusetts Institute of Technology).

⁷ Mapeo objeto-relacional (Object Relational Mapping).

- Incluye el framework Twig, un poderoso motor de plantillas que separa el código PHP del HTML permitiendo una amplia gama de posibilidades y además un extraordinario orden para el código del proyecto.

Se elige Symfony como framework a utilizar pues este reutiliza conceptos y desarrollos exitosos de terceros integrándolos como librerías para ser utilizados en cualquier momento, de esta forma se ahorra en tiempo. Además de la utilización de varios componentes incluidos en este framework que permiten una buena manipulación de los datos así como la seguridad de los mismos. Con Symfony se logra que una aplicación web tenga todo en su lugar, donde el mantenimiento y la corrección de errores se hace de una manera muy sencilla.

Un framework está formado por varias funcionalidades genéricas, las cuales son adaptadas por el desarrollador para dar solución a un problema en específico, para ello es necesario el uso de un Entorno de desarrollo integrado.

Entorno de desarrollo integrado (IDE)

Un 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.

Netbeans 8.0: Es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. [31]

- Está escrita en Java pero puede servir para cualquier otro lenguaje de programación.
- Es un producto libre y gratuito sin restricciones de uso. Soporta varios lenguajes, entre ellos: Java, C++, Ruby y PHP.
- Posee una instalación y actualización simple con un diseñador visual de formularios.
- Es fácilmente integrable con la herramienta CASE Visual Paradigm.

PhpStorm 8: Es un IDE potente para desarrollo PHP que permite simplificar procesos. [32]

- Su editor de PHP entiende perfectamente su estructura y soporta las versiones 5.3, 5.4, 5.5 y 5.6 de PHP.
- Proporciona finalización inteligente de código, comprobación de errores al instante o mezcla de lenguajes.

El IDE a utilizar es el Netbeans en su versión 8.0 dado que este proporciona al desarrollador un entorno adaptable a las características del lenguaje utilizado para la implementación de la

herramienta. Además facilita la creación de clases al integrarse con la herramienta Visual Paradigm que realiza el modelado de las mismas en primer lugar.

El almacenamiento de la información es de suma importancia durante el desarrollo de software, lo cual permite la manipulación y visualización de la misma. Este proceso se realiza a través de los gestores de base de datos, a continuación se presentan algunos de ellos.

Gestor de Base de Datos

Un Sistema Gestor de base de datos (SGBD) es un conjunto de programas que permiten crear y mantener una Base de datos, asegurando su integridad, confidencialidad y disponibilidad. Por tanto debe permitir:

- Definir una base de datos: Especificar tipos, estructuras y restricciones de datos.
- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.
- Manipular la base de datos: Realizar consultas, actualizarla, generar informes.

Algunas de las características deseables en un SGBD son: [33]

- Control de la redundancia: La redundancia de datos tiene varios efectos negativos (duplicar el trabajo al actualizar, desperdicia espacio en disco, puede provocar inconsistencia de datos) aunque a veces es deseable por cuestiones de rendimiento y seguridad.
- Restricción de los accesos no autorizados: Cada usuario ha de tener unos permisos de acceso y autorización.
- Cumplimiento de las restricciones de integridad: El SGBD ha de ofrecer recursos para definir y garantizar el cumplimiento de las restricciones de integridad.

Dentro de los SGBD se encuentran los que a continuación se refieren:

MySQL: Es un SGBD muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales contar con un alto grado de estabilidad y un rápido desarrollo. MySQL está disponible para múltiples plataformas. [34]

PostgreSQL 9.1.1: Es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS). Este está ampliamente considerado como el sistema de base 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. [35]

- 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.

El autor considera a partir de lo expuesto anteriormente que para la gestión y manejo de los datos relacionados con la presente investigación, el SGBD PostgreSQL 9.1.1 es el más indicado para la solución propuesta.

Para realizar la publicación de la herramienta y que la información sea mostrada al desarrollador en tiempo de ejecución, se hace uso de los servidores web. A continuación se describen algunos de ellos.

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. Entre los servidores web más usados se encuentran:

WampServer: Es un entorno de desarrollo web para Windows con el que se puede crear aplicaciones web con Apache, PHP y bases de datos MySQL. Incluye PHPMyAdmin y SQLiteManager para manejar las base de datos de una forma muy fácil. [36]

- WampServer es completamente gratuito.
- WAMP incluye, además de las últimas versiones de Apache, PHP Y MySQL, versiones anteriores de las mismas, para el caso de que se quiera testear en un entorno de desarrollo particular.

Apache Tomcat: Es una aplicación de código abierto. [26]

- Es desarrollado en un entorno abierto y participativo.
- 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.
- Su extensibilidad y construcción modular permite crear módulos para ampliar su funcionalidad.

- Ocupa muy poco espacio, teniendo su código binario en un tamaño total de apenas un megabyte, de modo que no es raro que se ejecute tan deprisa.
- Apache es el servidor web de aplicaciones más utilizado en la actualidad.

El autor considera a partir de lo expuesto anteriormente que el servidor web Apache Tomcat es el indicado para lograr un efectivo desarrollo y publicación de la herramienta a implementar debido a las características destacadas y su robustez que proporciona un alto grado de estabilidad.

Para que el tratamiento de la información se realice de forma visual y que el usuario interactúe con la herramienta publicada se requiere de un navegador web.

Navegador

Un navegador web (del inglés, web browser) es una aplicación de software que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web de todo el mundo a través de Internet. La funcionalidad básica de un navegador web es permitir la visualización de documentos de texto, posiblemente con recursos multimedia incrustados. Los documentos pueden estar ubicados en la computadora en donde está el usuario, pero también pueden estar en cualquier otro dispositivo que esté conectado a la computadora del usuario o a través de Internet, y que tenga los recursos necesarios para la transmisión de los documentos (un software servidor web). Tales documentos, comúnmente denominados páginas web, poseen hipervínculos que enlazan una porción de texto o una imagen a otro documento, normalmente relacionado con el texto o la imagen. [37]

Internet Explorer: Utiliza seguridad basada en zonas y grupos de sitios sobre determinadas condiciones, así como un usuario en la lista blanca. Ha sido objeto de muchas vulnerabilidades de seguridad y preocupaciones, a veces se requiere solo la visualización de una página web maliciosa para instalar el virus. Es un complemento indispensable y por defecto del sistema operativo Windows, gracias al cual ha logrado su alto nivel de uso, sin embargo debido a algunos fallos de seguridad que presenta, la confiabilidad hacia este navegador ha disminuido. [38]

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 y la seguridad. 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.

El autor para visualizar la herramienta a implementar utiliza el navegador Firefox 35.0.1 pues este cuenta con complementos para los desarrolladores, por ejemplo Firebug para seguir de cerca el

flujo de los datos que son entrados y procesados, de esta forma se tendrá completa visualización de lo que ocurre en la interfaz relacionada con el negocio.

Para agilizar el proceso de implementación de la herramienta, se harán uso de soluciones referentes a la arquitectura y diseño de la misma, denominados patrones de arquitectura y de diseño, respectivamente. A continuación se resumen los aplicados en la presente investigación.

1.6 Patrón de arquitectura

El influyente grupo de autores Buschmann, Meunier, Rohnert, Sommerlad y Stal [39] denominan un patrón de arquitectura como entidades que, con un empaquetado un poco distinto, no son otra cosa que los estilos. Estos patrones expresan esquemas de organización estructural fundamentales para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen guías y lineamientos para organizar las relaciones entre ellos.

La presente investigación se enfoca en el uso del patrón arquitectónico Modelo-Vista-Controlador (MVC) referido a los sistemas interactivos.

El patrón MVC surge con el objetivo de reducir el esfuerzo de programación, necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos, a partir de estandarizar el diseño de las aplicaciones. Es un paradigma que divide las partes que conforman una aplicación en el Modelo, las Vistas y los Controladores, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo. A partir del uso de frameworks basados en el patrón MVC se puede lograr una mejor organización del trabajo y mayor especialización de los desarrolladores y diseñadores. [40]

Modelo: Es la parte del patrón que se encarga del manejo de los datos y sus transformaciones. Es el propio sistema el encargado de relacionarlo con las Vistas y los Controladores.

Vista: Visualiza los datos manejados por el Modelo al usuario interactuando preferentemente con el Controlador.

Controlador: Actúa sobre los datos representados por el Modelo y centra toda la interacción entre la Vista y este. Gestiona los cambios en la información del Modelo y en las alteraciones que sufra la Vista.

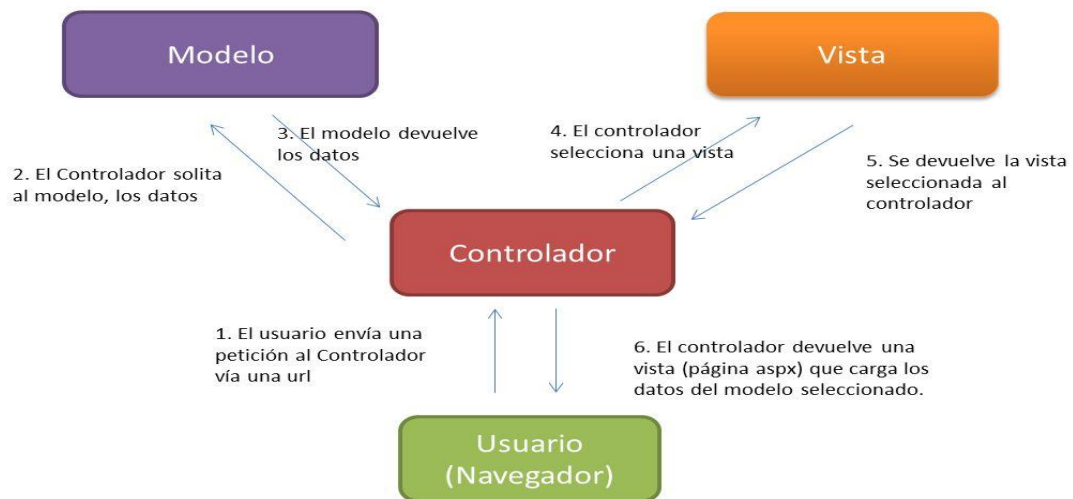


Figura 2 Estructura del patrón MVC. [40]

1.7 Patrones de diseño

Los patrones de diseño constituyen una herramienta fundamental para dar solución a problemas frecuentes en cualquier ámbito de desarrollo de software puesto que son el esqueleto de las soluciones referentes al diseño de interacciones e interfaces, permitiendo al usuario desarrollar una solución más fortalecida a partir de una guía predefinida, sin embargo un modelo no se puede construir sólo con la combinación de patrones, en dependencia de la necesidad se buscará el patrón ideal.

Según Rojas [41] los patrones de diseño son:

- Principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de software.
- Descripción de un problema y la solución a la que le da el nombre y que se puede aplicar en nuevos contextos.
- Sugieren algo repetitivo, no expresan nuevas ideas de diseño.

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.

Los patrones de diseño **GOF**⁸ se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

⁸ Gang of Four (Banda de los cuatro): Nombre con el que se conoce comúnmente a los autores del libro Design Patterns.

Patrones **creacionales**: Se encargan de las formas de crear instancias en los objetos. Su objetivo es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. Ejemplos de patrones de creación: Abstract Factory, Builder, Prototype, Singleton, Factory Method y Lazy Initialization. [42]

Patrones **estructurales**: Están relacionados y se aprecia cómo las clases y los objetos se combinan para formar nuevas estructuras más complejas y proporcionar nuevas funcionalidades. Ejemplos de patrones estructurales: Adapter, Bridge, Composite, Flyweight, Proxy, Decorator y Facade. [42]

Patrones de **comportamiento**: Están relacionados con algoritmos y asignación de responsabilidades a los objetos. Describen no solamente patrones de objetos o clases, sino también patrones de comunicación entre ellos. Ejemplos de patrones de comportamiento: Chain of Responsibility, Command, Interpreter, Iterator, Memento, Mediator, Observer, State, Strategy, Template Method y Visitor. [42]

Para la implementación de la herramienta se hará uso de los patrones GRASP: Creador, Controlador, Bajo Acoplamiento, Experto y Alta cohesión (restantes ver Anexo 2) y de los patrones GOF: Factory Method, Lazy Initialization, Adapter, Composite, Decorator y Template Method. La utilización de estos patrones va a permitir construir una herramienta más fácil de mantener, comprender y extender.

Una vez aplicados estos patrones, el proceso de implementación será guiado y definido por las buenas prácticas que estos ofrecen. Sin embargo se hace necesaria la realización de pruebas de software que validen la calidad de la herramienta resultante.

1.8 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 estado 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 mismo cumple con los requisitos. [43]

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. De acuerdo a lo que se desea examinar o verificar referente a la herramienta, pueden utilizarse varios métodos de pruebas.

Métodos de pruebas

Caja Blanca

Se conocen también como Prueba de Caja Transparente o de Cristal. Esta prueba consiste específicamente en cómo diseñar los casos de pruebas atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna, sin considerar los aspectos de rendimiento.

Dentro de la prueba de caja blanca se incluyen las técnicas de pruebas que serán descritas a continuación [2]:

- 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.

Para la presente investigación se aplica la técnica de prueba del camino básico.

Pruebas de caja negra

Se conocen también como prueba de caja opaca o inducida por los datos. Se centran en lo que se espera de un módulo, 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 [2]. En esencia permite encontrar:

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

Se aplica la técnica Partición de equivalencias para controlar las posibles ocurrencias de los elementos antes mencionados.

Pruebas de aceptación

Este tipo de prueba es realizada por el cliente sobre el sistema completo, evaluando el grado de calidad del software con relación a todos los aspectos relevantes, para que el uso del mismo se justifique. Se aplica la prueba beta en un entorno no controlado por el desarrollador para recibir por parte del cliente los problemas detectados.

1.9 Conclusiones parciales

En este capítulo se abordaron los elementos teóricos que sustentan la solución del problema:

- La complejidad de un RF se plantea como un problema donde es adecuado su tratamiento bajo incertidumbre. Para la implementación de la herramienta, teniendo en cuenta lo antes planteado, se hará uso de las definiciones utilizadas en el Método para determinar la complejidad de los RF de software.
- Con el estudio de diferentes herramientas que gestionan requisitos se pudo concluir que solo la implementada en Excel realiza el cálculo de la complejidad de los RF de software y que ninguna hace uso de técnicas de soft computing para realizarlo. Por ello se hace necesario el desarrollo de una herramienta que determine la complejidad.
- Las herramientas y tecnologías seleccionadas permitirán analizar, diseñar, implementar y validar la solución que se propone elaborar en los capítulos posteriores.

CAPÍTULO 2: REQUISITOS, ANÁLISIS Y DISEÑO

Introducción

En este capítulo se describe la propuesta de solución de la herramienta a desarrollar para el cálculo de la complejidad de los RF de software. Se presenta el modelo conceptual con las correspondientes descripciones de los conceptos asociados al dominio del problema. 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 RF y no funcionales. Se realiza el diagrama de clases del diseño y se muestra el Modelo Entidad-Relación (MER). También se explica la arquitectura y los principales patrones de diseño utilizados.

2.1 Propuesta de solución

La herramienta resultado de la presente investigación determina la complejidad de los RF de software, 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. La herramienta implementará el Método para determinar la complejidad de los RF de software resultante de una tesis de maestría de igual título defendida en el año 2014. [3]

Siguiendo las políticas de seguridad informática de la UCI, la herramienta contará con una previa autenticación de usuarios. De acuerdo a los permisos que estos posean, interactuarán con las diferentes funcionalidades, fundamentalmente la gestión de los RF de software a los cuales se les realizará el cálculo de la complejidad a partir de los criterios dados por los expertos.

El desarrollo de la herramienta parte de lograr una abstracción de la realidad referente al problema en cuestión, definiendo los principales conceptos que intervienen. Para esto se realiza el modelo conceptual.

2.2 Modelo conceptual

El Modelo conceptual explica cuáles son y cómo se relacionan entre sí los conceptos relevantes en un problema determinado. En la Figura 3 se muestra el modelo conceptual referente a la presente investigación.

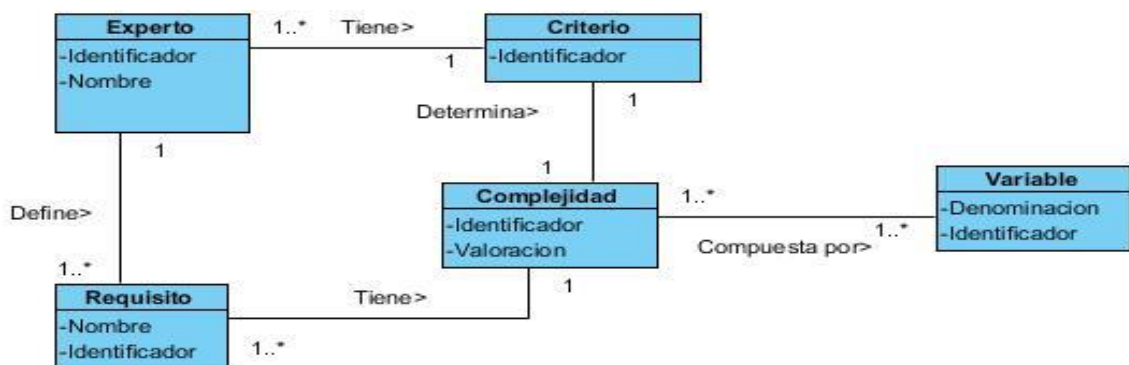


Figura 3 Modelo conceptual (elaboración propia).

Para una mejor comprensión, a continuación se describen cada uno de los conceptos que intervienen en el modelo conceptual:

Experto: Representa al usuario que a partir de sus conocimientos es capaz de emitir criterios sobre los requisitos a evaluar.

Criterio: Representa la valoración que emitirán los expertos a partir de sus consideraciones sobre los requisitos a evaluar.

Complejidad: Representa la característica que describe la dificultad de diseño e implementación de los requisitos dentro del proceso de desarrollo de software.

Variable: Representa los indicadores que aportan complejidad a los requisitos a evaluar por los expertos.

Requisito: Representa una condición o capacidad que la herramienta debe cumplir o tener para resolver un problema o alcanzar un objetivo.

Luego de comprender conceptualmente cómo debe funcionar la herramienta a desarrollar, se hace necesario realizar la captura de los RF y no funcionales que la materializarán en términos de implementación.

2.3 Requisitos

Técnicas para la captura de requisitos

Para la identificación de los RF y no funcionales de software se aplicaron las técnicas entrevista y tormenta de ideas, mencionadas en el capítulo anterior, ambas realizadas con analistas de proyectos de la UCI, específicamente del Centro de Informatización de Entidades (CEIGE) y con especialistas de la Dirección de Calidad UCI.

La herramienta cumple con los siguientes RF:

RF 1 Adicionar experto

RF 2 Modificar experto

RF 3 Eliminar experto

RF 4 Buscar experto

RF 5 Listar experto

RF 6 Importar requisito

RF 7 Adicionar requisito

RF 8 Modificar requisito

RF 9 Eliminar requisito

RF 10 Buscar requisito

RF 11 Listar requisito

RF 12 Establecer criterio

RF 13 Modificar criterio

RF 14 Listar criterio

RF 15 Eliminar criterio

RF 16 Calcular complejidad

RF 17 Exportar

RF 18 Autenticar

Una vez identificados los RF se evalúa la complejidad de los mismos para definir la dificultad de diseño e implementación en los que cada uno incurre.

Complejidad de los RF

La complejidad de los RF fue calculada a partir de la herramienta actual que forma parte del producto de trabajo Evaluación de requisitos. A partir de un conjunto de variables definidas se evalúa el grado de incidencia de las mismas en el requisito analizado. A continuación se muestra el resultado obtenido de la evaluación:

Requisitos/Complejidad	Alta	Media	Baja
RF 1	x		
RF 2	x		
RF 3		x	
RF 4			x
RF 5			x
RF 6	x		
RF 7	x		
RF 8	x		
RF 9		x	
RF 10			x
RF 11			x
RF 12	x		
RF 13	x		
RF 14			x
RF 15		x	
RF 16	x		
RF 17		x	
RF18		x	

Tabla 2 Complejidad de RF (elaboración propia).

Descripción de los RF de software

Especificación de Requisito Importar requisito

Descripción textual del requisito

Precondiciones	El usuario está autenticado en la herramienta y tiene permisos para realizar la acción.	
Flujo de eventos		
Flujo básico Importar requisito		
1.	El usuario presiona la opción Importar.	
2.	Se muestra una interfaz para escoger el fichero que se desea importar.	
3.	Se selecciona el archivo a importar: producto de trabajo Criterios para validar requisitos del producto.	
4.	El usuario especifica el número de la hoja del Excel importado (pestaña), Columna, Fila inicial, Fila final.	
5.	El usuario presiona la opción Aceptar.	
6.	Se importan a la herramienta los requisitos.	
Pos-condiciones		
1.	Se ha importado un conjunto de requisitos.	
Flujos alternativos		
Flujo alternativo 1.a Importar requisitos desde un archivo inválido		
1	El usuario presiona la opción Aceptar.	
2	Se muestra una alerta indicando que el archivo seleccionado no es el correcto.	
3	Volver al flujo básico 2.	
Pos-condiciones		
1.	N/A	
Flujos alternativos		
Flujo alternativo 1. b Importar requisitos dejando campos vacíos		
1.	El usuario presiona la opción Aceptar.	
2.	Se muestra en los campos en blanco una alerta indicando que el campo es obligatorio.	
3.	Se presiona la opción Aceptar.	
4.	Volver al flujo básico 4.	
Validaciones		
1.	Ver Modelo_conceptual-1	
Conceptos	Requisito	Visibles en la interfaz: <ul style="list-style-type: none">• Hoja, Fila inicial, Columna, Fila final Utilizados internamente: <ul style="list-style-type: none">• id
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz gráfica de usuario

Inicio > Importar desde archivo

Importar requisitos

Información

LOS REQUISITOS REGISTRADOS CUYOS NOMBRES COINCIDAN CON LOS QUE DESEA IMPORTAR DESDE EL ARCHIVO, NO SERÁN SOBRESCRITOS!!!

Hoja	<input type="text"/>	Columna	<input type="text"/>
Fila inicial	<input type="text"/>	Fila final	<input type="text"/>
Archivo	<input type="button" value="Examinar..."/> No se ha seleccionado ningún archivo.		

Formatos de entrada/salida

NA

Entradas

NA

Salidas

NA

Especificación de Requisito Calcular complejidad

Descripción textual del requisito

Precondiciones	El usuario está autenticado en la herramienta y tiene permisos a realizar la acción. El requisito a evaluar la complejidad debe tener al menos un criterio de experto asociado.	
Flujo de eventos		
Flujo básico Calcular complejidad		
1.	El usuario selecciona el requisito al que le desea calcular la complejidad.	
2.	El usuario presiona la opción Calcular complejidad.	
3.	Se muestra una interfaz con todos los criterios realizados por los expertos sobre las variables asociadas al requisito: <ul style="list-style-type: none">• Experto• Variables.• Valoración• Nivel de importancia asociado a cada variable	
4.	El usuario presiona la opción Aceptar.	
5.	Se cierra la interfaz para el cálculo.	
Pos-condiciones		
1.	Se ha calculado la complejidad del requisito funcional.	
2.	Se actualiza la columna Complejidad del listado de requisitos.	
Flujos alternativos		
Flujo alternativo *.a Cancelar acción		
1.	El usuario presiona la opción Cancelar.	
Pos-condiciones		
1.	Se cierra la interfaz.	
Validaciones		
1.	Ver Modelo_conceptual-1	
Conceptos	Requisito	Visibles en la interfaz: <ul style="list-style-type: none">• Experto• Variables• Valoración• Nivel de importancia Utilizados internamente: <ul style="list-style-type: none">• Id
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz gráfica de usuario

Inicio > Requisitos > Calcular complejidad

Calcular complejidad

Requisito: Adicionar centro rector padre

Expertos	Yarenis Echemendia González		Noidis Barroso Hidalgo	
Variable	Valoración	NI	Valoración	NI
Interfaces	Media	5	Media	2
Facilidad de cambio	Alta	5	Alta	4
Dependencia con otros requisitos	Alta	5	Alta	4
Requisitos asociados	Baja	5	Media	4
Transacciones	Alta	4	Baja	2
Diferentes comportamientos	Media	5	Alta	5
Restricciones	Alta	4	Media	3

Calcular complejidad

Formatos de entrada/salida

NA

Entradas

NA

Salidas

NA

Requisitos no funcionales de software (RNF)

Requisitos de Usabilidad

RNF 1 Para cada acción correcta que se realice en la herramienta, se muestra un mensaje de información en correspondencia con la acción realizada. Estos mensajes se ubican en la esquina superior derecha y son de color azul claro.

RNF 2 Para cada acción incorrecta que se realice en la herramienta, se muestra un mensaje de información en correspondencia con la acción realizada. Estos mensajes se ubican en la esquina superior derecha y son de color amarillo claro.

RNF 3 Todos los campos que son obligatorios están validados, cuando se intenta dejar alguno en blanco, debajo del propio campo se muestra una alerta indicando la obligatoriedad del mismo.

RNF 4 La iconografía utilizada es única para cada operación, permitiendo representar todos los conceptos del dominio de la herramienta con un ícono distintivo. Ejemplo: Las acciones de inserción son de color azul, las de edición son de color verde y las de eliminación tienen color rojo. En cada una de las interfaces que contienen acciones de Adicionar, Editar y Eliminar; estas aparecen en el mismo orden.

RNF 5 El idioma de todas las interfaces de la herramienta está en español.

RNF 6 El orden de desplazamiento por campos en cada formulario de la herramienta siempre será de izquierda a derecha.

RNF 7 La confirmación de la entrada de datos en cada formulario de la herramienta puede hacerse mediante el uso de mouse y/o teclado.

RNF 8 En el escenario en el que se listan los RF se permite ajustar el número máximo de los registros a mostrar, siendo el mínimo de 5 y el máximo de 60 por páginas.

RNF 9 En la herramienta no existe una cadena de más de tres interfaces de usuario para lograr una funcionalidad completa.

Requisitos de Seguridad

RNF 1 La información sensible sólo es vista por los usuarios con el nivel de acceso adecuado, mostrándose las funcionalidades de la herramienta de acuerdo al usuario que esté activo.

RNF 2 La autenticación es la primera acción del usuario en la herramienta y consistirá en proveer un nombre de usuario único y una contraseña.

RNF 3 La herramienta estará disponible para su utilización las 8 horas de los 24 días laborables del mes.

Requisitos de Restricciones en el diseño y la implementación

RNF 1 Los componentes de la herramienta son desarrollados siguiendo el principio de alta cohesión y bajo acoplamiento. La lógica de presentación es independiente de la lógica de negocio, centrandose su función en la interfaz de usuario y validaciones de los datos de entrada.

Requisitos de Escalabilidad

RNF 1 La herramienta tiene la capacidad de permitir en el futuro el desarrollo de nuevas funcionalidades asociadas a la inserción o eliminación de variables que otorguen complejidad a los RF. Para ello existe un panel de administración en desarrollo con todos los nomencladores que se utilizan en la herramienta.

Requisitos de Portabilidad

RNF 1 La herramienta posee la capacidad de ser adaptada a los ambientes especificados y de coexistir con otro software independiente en un ambiente común.

Requisitos de Mantenibilidad

RNF 1 La herramienta posee la capacidad para permitir la aplicación de una modificación especificada. La modificación interna de un componente de la herramienta no afecta a las funcionalidades o componentes que dependan de esta.

Requisitos de software

RNF 1 Cliente: Instalar un navegador web que soporte JavaScript. Se recomienda Mozilla Firefox 3.5 o superior.

RNF 2 Servidor: Instalar Apache Tomcat como servidor web y PostgreSQL como gestor de base de datos. Instalar el Sistema Operativo: GNU Linux, Windows XP o superior.

Requisitos de hardware

RNF 1 Cliente: Procesador Pentium o superior, 256 Mb de RAM.

RNF 2 Servidor de aplicaciones: Capacidad de disco duro superior a 80 GB, microprocesador Pentium superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

RNF 3 Servidor de base de datos: Capacidad de disco duro superior a 180 GB, microprocesador Pentium superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

Técnicas para la validación de los RF de software

Para validar los requisitos identificados se aplicó la técnica de Prototipos de Interfaz de Usuario, se hicieron simulaciones del posible producto. Cada uno de los prototipos le permitió a los especialistas tener una idea de cada una de las interfaces de la herramienta. Estos se realizaron de forma no funcional con la herramienta Visual Paradigm 8.0 para UML, logrando una mayor aprobación por parte del usuario final.

En aras de refinar los RF y traducirlos en términos de implementación se realizan los diagramas de clases de diseño.

2.4 Clases del diseño

Los diagramas de clases son muy utilizados en el modelado de sistemas, 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. [22] A continuación se muestra el diagrama de clases del diseño representado en tres partes: Vista, Controlador y Modelo para una mayor comprensión del mismo (ver diagrama completo Anexo 3):

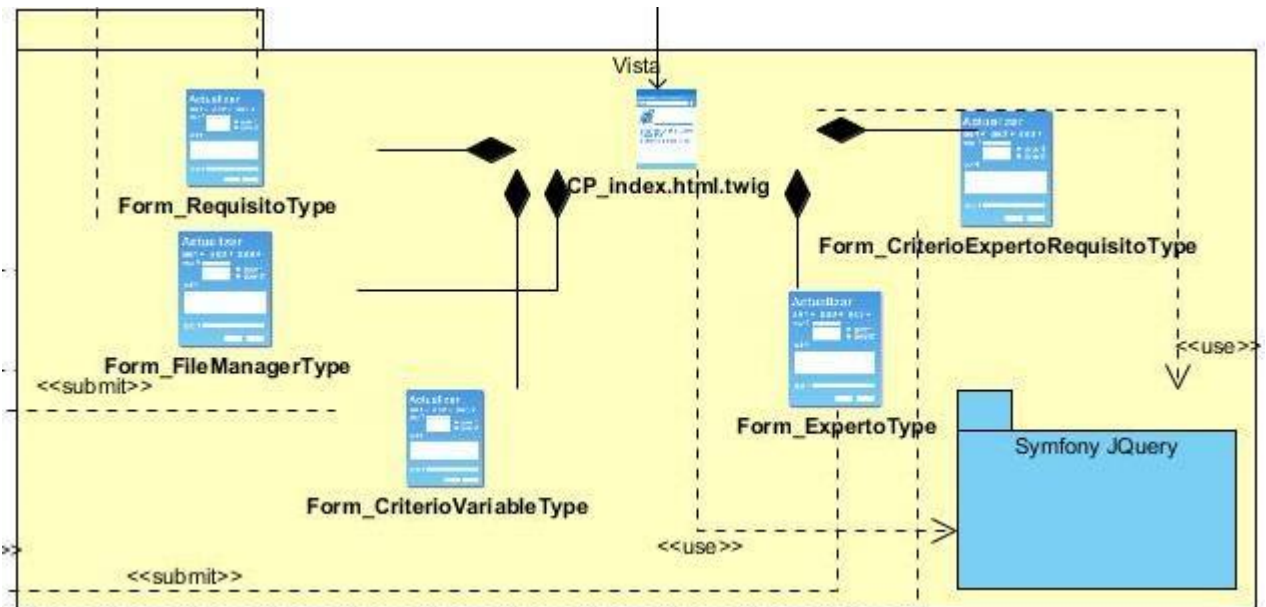


Figura 4 Paquete Vista (elaboración propia).

Clases	Descripción
Form_RequisitoType	Representa el formulario de la entidad Requisito a crear o modificar.
Form_CriterioExpertoRequisitoType	Representa el formulario de la entidad CriterioExpertoRequisito a crear o modificar.
Form_CriterioVariableType	Representa el formulario de la entidad CriterioVariable a crear o modificar.
Form_ExpertoType	Representa el formulario de la entidad Experto a crear o modificar.
Form_FileManagerType	Representa el formulario de la entidad FileManager a crear.

Tabla 3 Descripción del diseño de clases paquete Vista (elaboración propia).

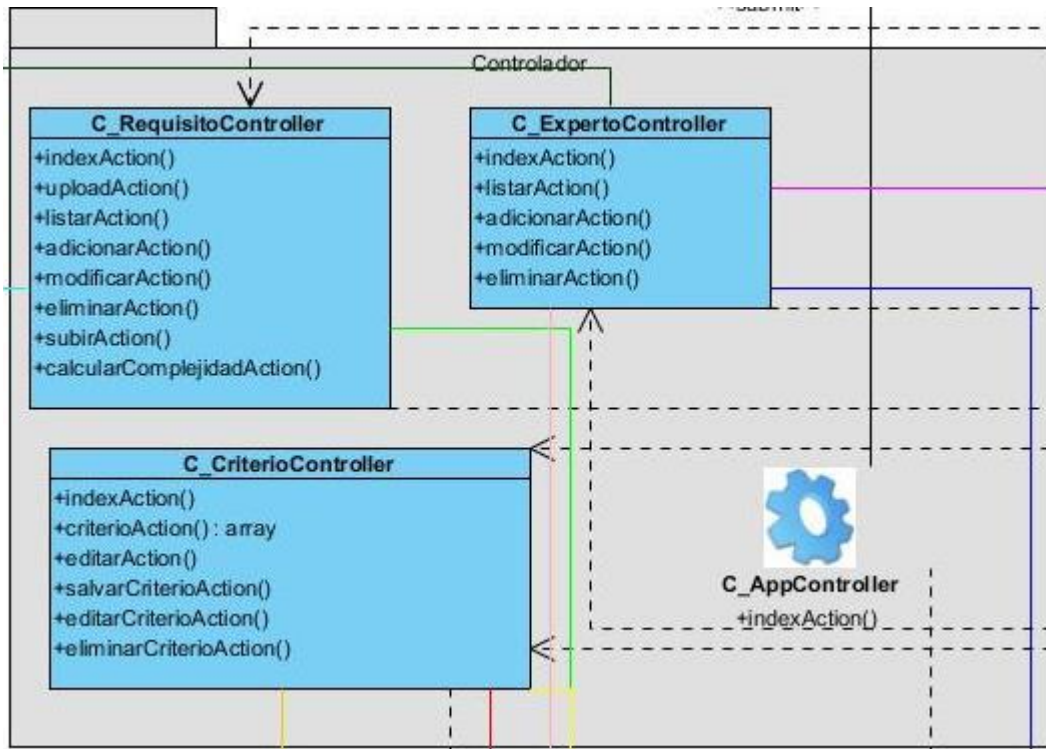


Figura 5 Paquete Controlador (elaboración propia).

Clases	Descripción
C_ExpertoController	Gestiona el comportamiento de las entidades Experto y CriterioExpertoRequisito, así como el flujo de datos de los formularios asociados a las mismas representados por las vistas.
C_RequisitoController	Gestiona el comportamiento de la entidad Requisito así como el flujo de datos de los formularios asociados a la misma representados por las vistas.
C_CriterioController	Gestiona el comportamiento de la entidad CriterioVariable así como el flujo de datos de los formularios asociados a la misma representados por las vistas.
C_AppController	Construye y renderiza la plantilla index.htm.twig de la cual heredan todos los formularios.

Tabla 4 Descripción del diseño de clases paquete Controlador (elaboración propia).

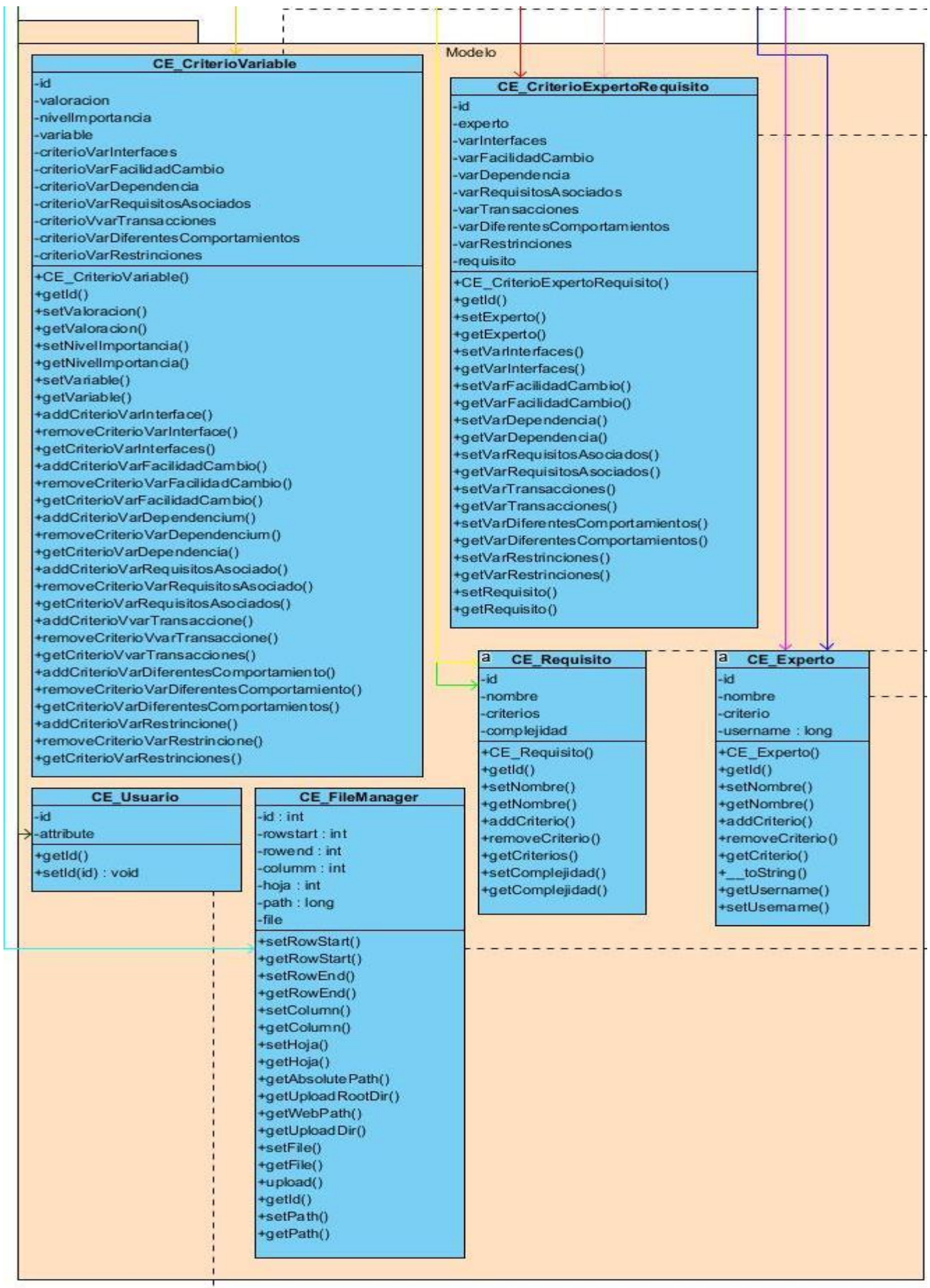


Figura 6 Paquete Modelo (elaboración propia).

Clases	Descripción
CE_CriterioVariable	Representación de la entidad CriterioVariable y sus correspondientes atributos.
CE_CriterioExpertoRequisito	Representación de la entidad CriterioExpertoRequisito y sus correspondientes atributos.
CE_Experto	Representación de la entidad Experto y sus correspondientes atributos.
CE_Requisito	Representación de la entidad Requisito y sus correspondientes atributos.
CE_Usuario	Representación de la entidad Usuario y sus correspondientes atributos.
CE_FileManager	Representación de la entidad FileManager y sus correspondientes atributos.

Tabla 5 Descripción del diseño de clases paquete Modelo (elaboración propia).

2.1 Diagramas de secuencia

Un diagrama de secuencia muestra una interacción, que representa la secuencia de mensajes entre las instancias de clases, componentes, subsistemas o actores, así como instancias y eventos de ejemplo, en lugar de clases y métodos; más de una instancia del mismo tipo puede aparecer en el diagrama y más de una aparición del mismo mensaje también puede aparecer. [44]

A continuación se muestran los diagramas de secuencia de los RF: Importar requisitos y Calcular complejidad.

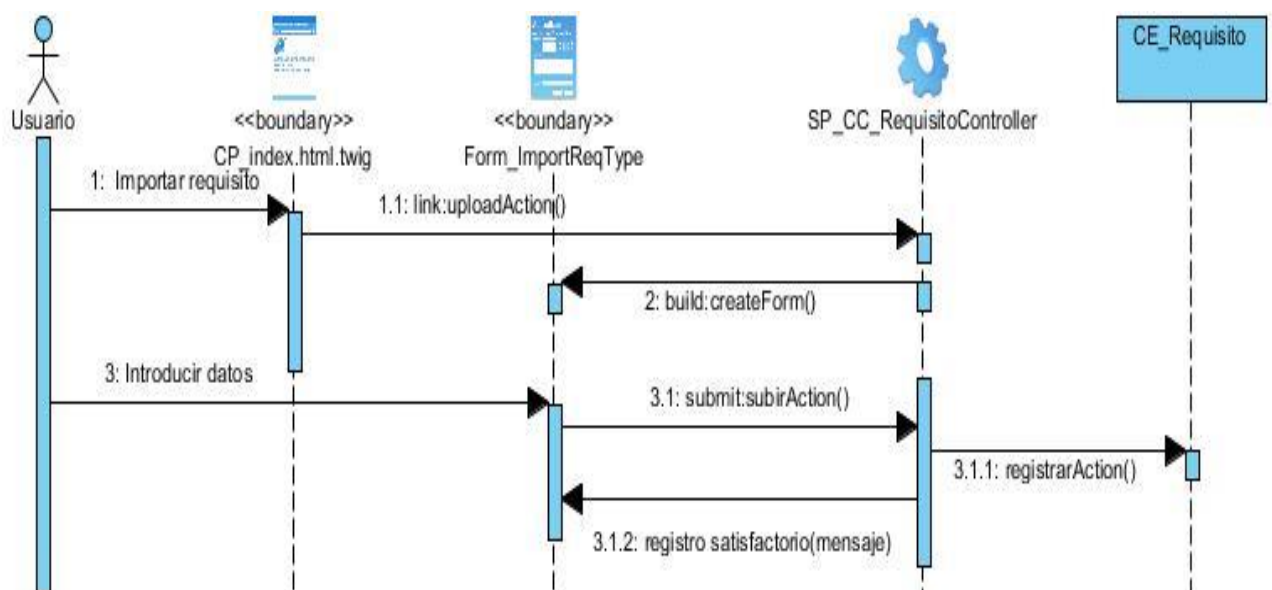


Figura 7 Diagrama de secuencia del Gestionar Requisitos Escenario: Importar Requisito (elaboración propia).

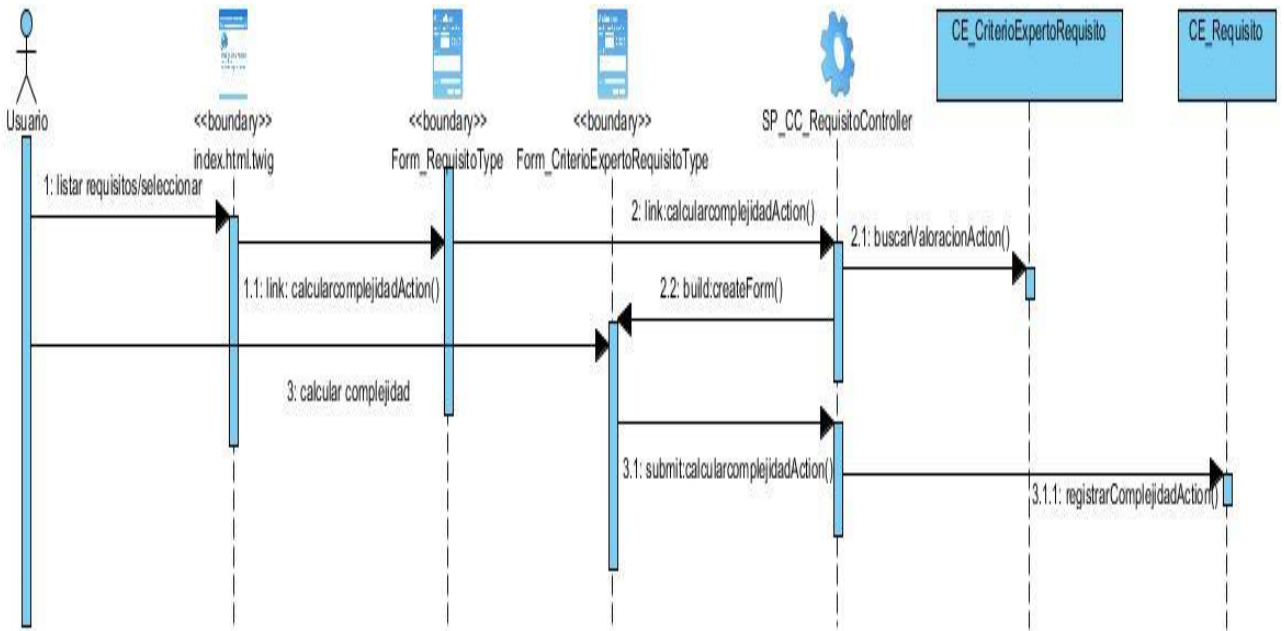


Figura 8 Diagrama de secuencia del Gestionar Complejidad Escenario: Calcular complejidad (elaboración propia).

La información es un factor determinante en la construcción de un software, el tratamiento de la misma requiere del modelado de su estructura para conocer cómo persistirá en el tiempo. Esta representación se realiza a través de los modelos de datos.

2.2 Modelo de datos

El Modelo Entidad Relación (MER) es un tipo de diagrama para modelado de bases de datos. Los modelos de datos comprenden aspectos relacionados con: estructuras y tipos de datos, operaciones y restricciones. [45] Su objetivo es representar relaciones que existen en la vida real entendiendo su semántica. A continuación se muestra el modelo de datos de la herramienta para determinar la complejidad de los RF de software, está compuesto por nueve entidades relacionadas entre sí. Ver Figura 9:

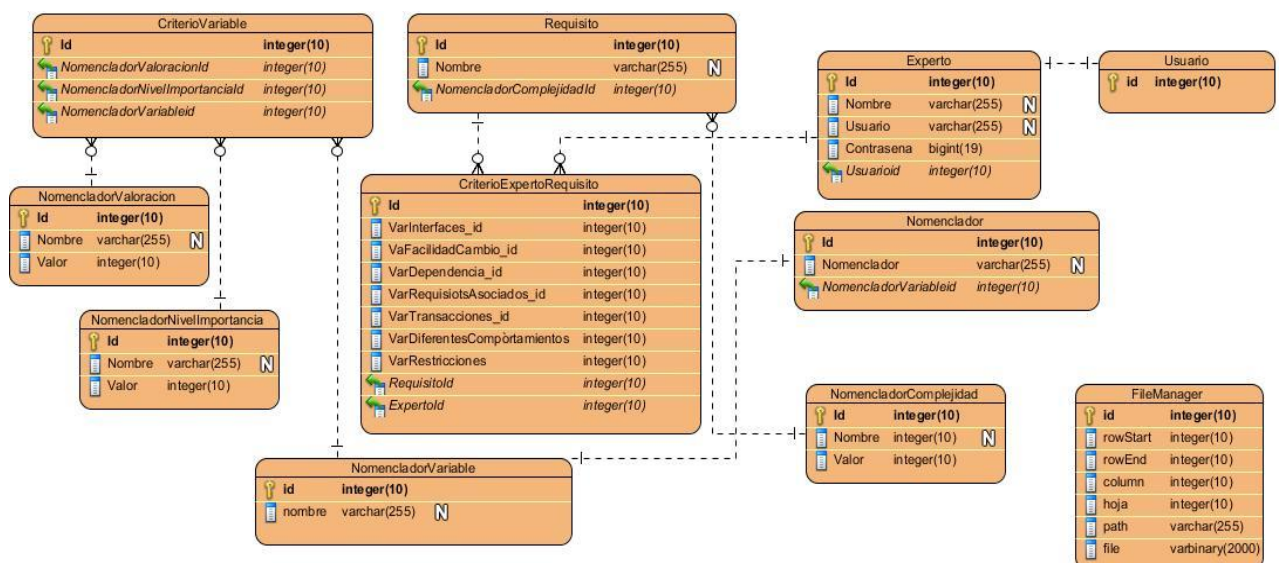


Figura 9 Modelo de datos (elaboración propia).

A continuación se muestran algunas descripciones de las tablas generadas por el modelo de datos del sistema:

Nombre: "Requisito"		
Descripción: Almacena los datos correspondientes a cada requisito registrado.		
Atributo	Tipo	Descripción
id	Integer	Identificador de la tabla.
Nombre	Varchar (255)	Nombre del requisito registrado.
NomencladorComplejidadId	Integer	Identificador de la tabla NomencladorComplejidad (Llave foránea).

Tabla 6 Diccionario de datos tabla Requisito (elaboración propia).

Nombre: "Experto"		
Descripción: Almacena los datos correspondientes a cada requisito registrado.		
Atributo	Tipo	Descripción
id	Integer	Identificador de la tabla.
Nombre	Varchar (255)	Nombre del experto registrado.
Usuario	Varchar (255)	Usuario asignado al experto.
Contraseña	Text	Contraseña asignada al experto.
Usuarioid	Integer	Identificador de la tabla Usuario (Llave foránea).

Tabla 7 Diccionario de datos tabla Experto (elaboración propia).

2.3 Patrones de diseño utilizados

Los principios del diseño necesarios para construir un software pueden codificarse, explicarse y aplicarse de modo metódico utilizando los denominados patrones de diseño. Cada patrón trata un problema específico, que se repite en el diseño o implementación de un software. Un patrón no es más que una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos, en otras palabras, "los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software". [46]

Estos son considerados soluciones que aplican ciertos estilos que ayudan a la creación de software. En el caso de los patrones **GRASP**, estos enfocan su aplicación en la asignación de responsabilidades dentro de la creación de un software.

- **Patrón Experto**

Es uno de los patrones que más se utiliza cuando se trabaja con Symfony, con la inclusión de la librería Doctrine para mapear la base de datos. Symfony utiliza esta librería para realizar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades. Las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla que representan.

- **Patrón Alta Cohesión**

La característica principal de este patrón es asignar responsabilidades de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase. Symfony permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. Un ejemplo de ello, es la clase `C_CriterioExpertoRequisitoController`, la cual está formada por varias funcionalidades para colaborar con otras clases en la realización de diferentes operaciones.

- **Patrón Bajo Acoplamiento**

La característica principal de este patrón es mantener las clases más independientes entre sí y con la menor cantidad de relaciones; lo cual posibilita que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases. Esto potencia la reutilización y disminuye la dependencia entre las clases, asignándoles una responsabilidad para mantener un bajo acoplamiento. Para alcanzar un bajo acoplamiento en la solución, las clases que implementan la lógica del negocio no poseen ninguna asociación entre ellas, lo que proporciona que la dependencia en este caso sea baja.

- **Patrón Creador**

La característica principal de este patrón es que permite identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Asignarle a la clase B la responsabilidad de crear una instancia de la clase A. Este patrón se puede observar al realizar las inserciones en la base de datos con el uso de los formularios de Symfony. La clase del documento que se va a insertar, es la responsable de crear el objeto del formulario correspondiente.

- **Patrón Controlador**

La aplicación de este patrón se evidencia en la clase `C_RequisitoController` el cual se utiliza para asignar la responsabilidad de controlar el flujo de eventos de la herramienta, a clases específicas.

Dentro de los patrones **GOF** se utilizaron: [47]

- **Factory Method**

Define una interfaz para la creación de un objeto, permitiendo a las subclasses decidir qué clase instanciar. Este patrón se utilizó en el componente Formulario de Symfony.

- **Lazy Initialization**

Este patrón se encarga de retrasar la creación de un objeto, el cálculo de algún valor u otro proceso costoso hasta la primera vez que es realmente necesario. Su uso permite optimizar el rendimiento, guardar consumo de memoria, establecer conexiones cuando realmente se necesita y la obtención de información bajo demanda. Este patrón se utilizó en el componente de Inyección de Dependencias de Symfony.

- **Adapter**

El patrón Adapter permite a la interfaz de una clase existente ser utilizada por cualquier otra interfaz. Esto permite que las clases trabajen entre sí sin tener que cambiar su código. Este patrón se utilizó en el componente de Seguridad de Symfony.

- **Composite**

Este patrón permite al cliente tratar objetos individuales y composiciones de objetos representando árboles de objetos uniformes. Este patrón puede ser verificado en el componente Formulario de Symfony.

- **Decorator**

Incorpora responsabilidades a objetos sin dividir sus clases en otras clases. Esto permite la extensión de objetos sin desbordamiento del código. Este patrón se utilizó en el componente HttpKernel de Symfony.

- **Template Method**

Permite a las subclasses redefinir ciertos pasos de un algoritmo sin cambiar la estructura del mismo. Este patrón se utilizó en el componente de Seguridad de Symfony.

2.4 Conclusiones parciales

Al finalizar el capítulo vale destacar:

- La creación del modelo conceptual sirvió como punto de partida para entender los principales conceptos relacionados con el desarrollo de la herramienta resultado de la presente investigación.

- A partir de las técnicas de captura de requisitos, entrevista y tormenta de ideas, se identificó un total de 18 RF.
- La validación de los RF con la aplicación de la técnica de Prototipo de Interfaz de Usuario, demostró que estos presentan las condiciones requeridas y están en correspondencia con las necesidades que debe cubrir la herramienta.
- El diagrama de clases del diseño permitió conocer la estructura interna y las relaciones entre las clases que cubren los RF identificados para lograr la implementación de estos.
- La realización del modelo de datos permitió conocer las relaciones existentes entre las tablas de la base de datos para su correcta manipulación.
- El patrón arquitectónico MVC permitió dividir las partes que conforman la aplicación en el modelo, las vistas y los controladores para estructurar la lógica interna del código y fomentar la reutilización del mismo.
- La utilización de los patrones de diseño GRASP y GOF, permitieron agilizar el proceso de implementación de la herramienta al aplicarlos en el diseño de la herramienta.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN

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. Se valida el diseño propuesto a través de las métricas Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC). Se describen las pruebas de caja negra y caja blanca realizadas a la herramienta y sus resultados, así como la validación de la herramienta en términos de negocio a través de la prueba de aceptación.

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. [22]

Dentro del modelo de implementación se encuentran los diagramas de componentes.

Diagrama de componentes

Un diagrama de componentes muestra las dependencias lógicas entre los componentes de software, sean estos elementos fuentes, binarios o ejecutables. Los diagramas de componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. [48]

A continuación se muestra el diagrama de componente de la herramienta. Ver Figura 10:

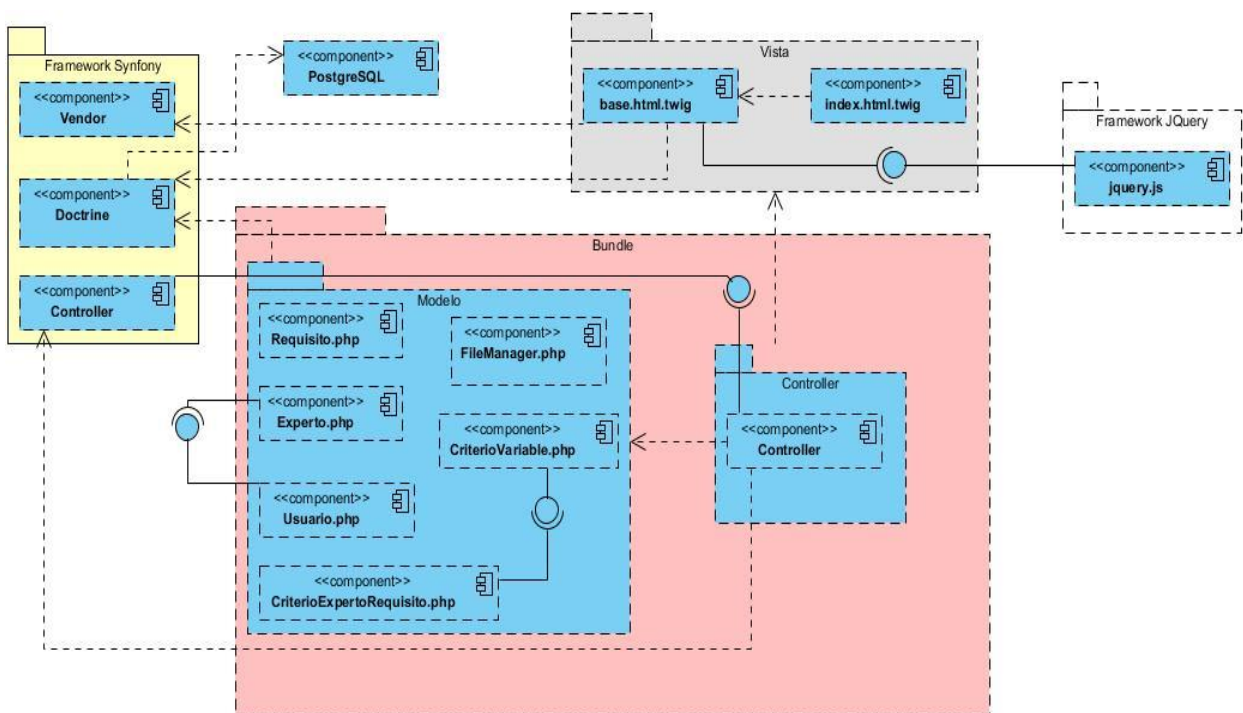


Figura 10 Diagrama de componentes (elaboración propia).

Para un mayor entendimiento, a continuación se describen cada uno de los componentes.

Symfony: Paquete que está integrado por tres componentes. El primero es el componente *Vendor* el cual contiene las bibliotecas necesarias para el desarrollo del sistema. El segundo es el componente *Controller* donde se encuentran todas las clases controladoras del sistema. Por último en este paquete se encuentra el componente *Doctrine* el cual se utiliza para el modelo y el acceso a la base de datos.

PostgreSQL: Representa a la base de datos del sistema.

Modelo: Contiene las clases responsables de manejar la información contenida en las tablas de la base de datos. Utiliza el componente *Doctrine* para acceder a la base de datos.

Vista: Aquí se encuentra el componente *index.htm.twig* representando a las vistas del sistema. Utiliza el componente *base.html.twig* que es la plantilla que se le aplica a todas las páginas haciendo uso del fichero *jquery.js*, definido dentro del paquete JQuery para conformar las vistas.

Controlador: Este paquete contiene el componente *Controller* donde se encuentran todas las clases controladoras del sistema.

JQuery: Este paquete contiene la biblioteca *jquery.js* utilizada para el trabajo con JavaScript.

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.

3.2 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. [49]

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. [50]

Nomenclatura de las clases

Cuando el nombre de la clase sea compuesto se empleará notación **PascalCasing**, la cual define que cada palabra iniciará con letra mayúscula y con solo leerlo se reconoce el propósito de la misma. Ejemplo: *RequisitoController*. En este caso el nombre de clase está compuesto por dos palabras iniciadas cada una con letra mayúscula, después de la primera.

- Nomenclatura según el tipo de clase:

Clases controladoras: Las clases controladoras después del nombre llevan la palabra: "Controller".

Ejemplo: RequisitoController.

Clases entidades: Las clases que representan las entidades se nombran de forma que se entienda claramente al objeto al que hacen referencia.

Ejemplo: Requisito.

Clases formularios: Las clases formularios después del nombre llevan la palabra: "Type".

Ejemplo: RequisitoType.

Nomenclatura de las funcionalidades y atributos

El nombre a emplear para las funcionalidades y los atributos se escribe con la inicial del identificador en minúscula, en caso de que sea un nombre compuesto se empleará también la notación **CamelCasing** que es similar a la antes mencionada **PascalCasing** con la excepción de que la primera letra es minúscula.

Ejemplo de un método: calcularComplejidadAction(). El nombre del método está compuesto por tres palabras, la primera en minúsculas y las restantes iniciadas con letra mayúscula.

En las principales funcionalidades de las clases controladoras se escribe el nombre y seguida la palabra: "Action".

Ejemplo de método: calcularComplejidadAction().

Ejemplo de atributo: idRequisito. El nombre del atributo está compuesto por dos palabras, la primera en minúscula y la restante iniciada con letra mayúscula.

Nomenclatura de los comentarios

Los comentarios deben ser lo suficientemente claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando. En la realización de un software se deben realizar comentarios en las funciones complejas para lograr una mejor comprensión del código y todo lo que se haga dentro del desarrollo. A continuación se muestra un ejemplo de los comentarios que se realizan en el código de la aplicación con el objetivo de lograr un código más legible y reutilizable y así se pueda aumentar su mantenibilidad a lo largo del tiempo.

```
<?php
/**
 * Creado por Alejandro el 01/03/2015.
 * Clase puente que contiene la funcionalidad calcularComplejidad.
 */

namespace App\CoreBundle\Bridge;

use ...

class AlgoritmoBridge
{
    private $doctrine ;
    //Acción que permite calcular la complejidad de un RF a partir de los criterios asociados al mismo.
    function calcularComplejidad(Requisito $requisito)//1
}
```

Figura 11 Ejemplo de comentarios en el código (elaboración propia).

Como parte de la construcción de la solución propuesta se hace necesaria la elaboración del diagrama de despliegue.

3.3 Diagrama de despliegue

El mismo describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos del cómputo [51]. Con el objetivo de representar la relación entre la arquitectura de software y la arquitectura de hardware de la herramienta para su correcto funcionamiento, en la Figura 12 se presenta el diagrama de despliegue.

El diagrama de despliegue realizado representa tres nodos principales. El nodo PC-Cliente que requiere de un navegador que soporte Java Script, el nodo Servidor-Web, en el cual debe estar instalado el servidor Apache Tomcat, en el nodo Servidor-BD debe estar instalado el SGBD PostgreSQL .

El nodo PC-Cliente estará conectado mediante el Protocolo de Transferencia de Hipertexto HTTP al nodo procesador que representa al Servidor-Web. La conexión entre el servidor web y el servidor de base de datos se realizará mediante el protocolo de comunicación TCP/IP.

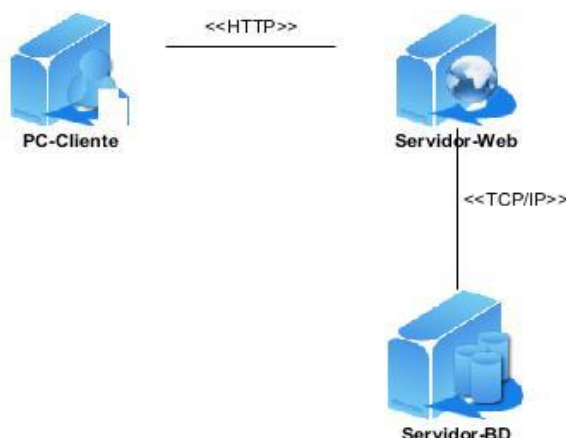


Figura 12 Diagrama de despliegue (elaboración propia).

Con el objetivo de verificar el estado del diseño de la herramienta a implementar se realiza su validación a través del uso de métricas y atributos de calidad.

3.4 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. [2]

Para la evaluación de la calidad del diseño propuesto para 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: [2]

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.

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 Tamaño operacional de clases TOC (elaboración propia).

Atributos	Categoría	Criterio
Responsabilidad	Baja	\leq Prom. (5.25)
	Media	Entre Prom. Y 2* Prom
	Alta	$>$ 2* Prom
Complejidad de implementación	Baja	\leq Prom
	Media	Entre Prom. y 2* Prom
	Alta	$>$ 2* Prom
Reutilización	Baja	$>$ 2* Prom
	Media	Entre Prom. y 2* Prom
	Alta	\leq Prom

Tabla 9 Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC (elaboración propia).

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 13:

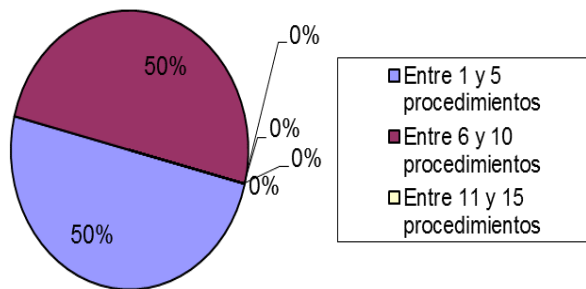


Figura 13 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad, ver Figura 14:

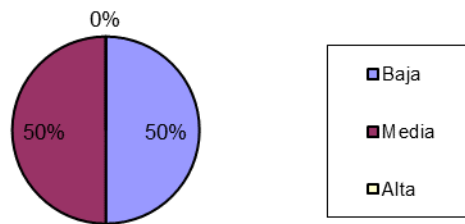


Figura 14 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación, ver Figura 15:

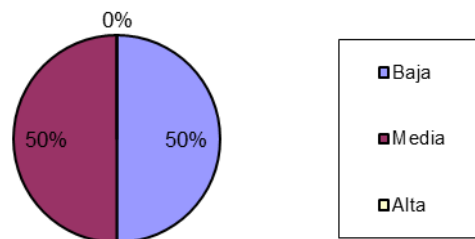


Figura 15 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización, ver **¡Error! No se encuentra el origen de la referencia.6:**

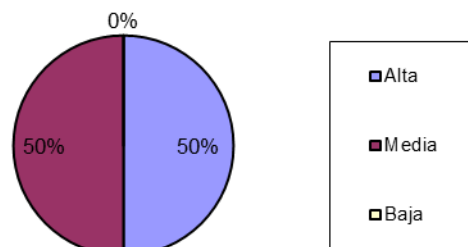


Figura 16 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización (elaboración propia).

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto para el sistema es simple y tiene una calidad aceptable, teniendo en cuenta que las clases se encuentran equilibradas en partes iguales (50%) en correspondencia con la cantidad de operaciones y la media registrada en las mediciones. Los atributos de calidad se encuentran en un nivel satisfactorio en el 50% 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.

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 aumenta en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica una disminución de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 10 Relaciones entre clases RC (elaboración propia).

Atributos	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de mantenimiento	Baja	\leq Prom (1.25)
	Media	Entre Prom. y 2* Prom
	Alta	$> 2^* Prom$
Reutilización	Baja	$> 2^* Prom$
	Media	Entre Prom. y 2* Prom
	Alta	$\leq Prom$
Cantidad de Pruebas	Baja	$\leq Prom$

	Media	Entre Prom. y 2* Prom
	Alta	> 2* Prom

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

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 17:

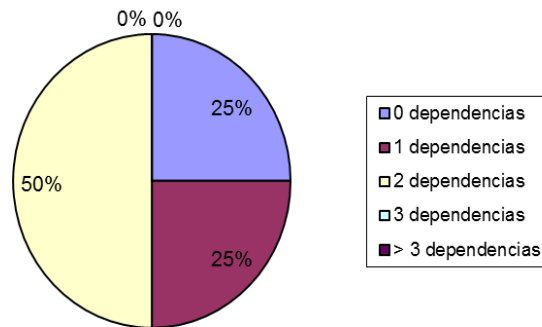


Figura 17 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento, ver Figura 18:

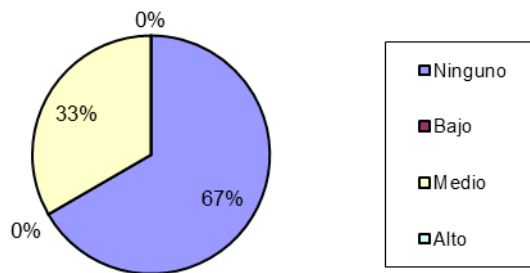


Figura 18 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento, ver Figura 19:

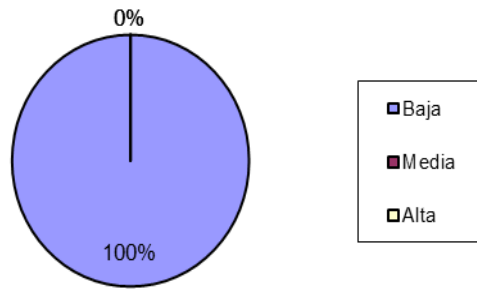


Figura 19 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de pruebas, ver Figura 20:

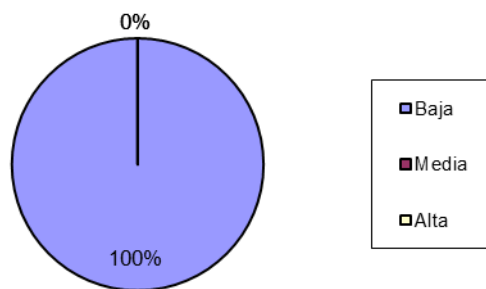


Figura 20 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de pruebas (elaboración propia).

Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización, ver Figura 21:

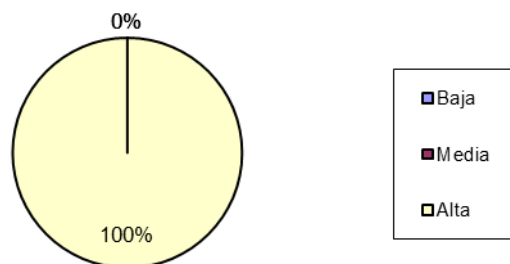


Figura 21 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización (elaboración propia).

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica RC, se puede concluir que el diseño propuesto para el sistema es simple y tiene una calidad aceptable. Los atributos de calidad se encuentran en un nivel satisfactorio, en el 100% 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 100% de las clases.

Una vez realizada la validación del diseño propuesto para la herramienta resultado de la presente investigación, se hace necesario validarla en términos de calidad, para esto se hace uso de las pruebas de software.

3.5 Pruebas de software aplicadas a la herramienta

Las pruebas de software no garantizan que un sistema esté libre de errores, sino que se detecten la mayor cantidad de defectos posibles para su debida corrección. Estas 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 componente. [52]

Prueba de Caja blanca: Técnica del Camino Básico

El camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que, durante la prueba, se ejecuta por lo menos una vez cada sentencia del programa. Con el uso del cálculo de la complejidad ciclomática se obtiene la cantidad de caminos que se deben buscar. [2]

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 calcularComplejidad (Requisito \$requisito) contenido en la clase AlgoritmoBridge.

```
function calcularComplejidad(Requisito $requisito) //1
{
    $pesos = 0 ; //1
    $influencia = 0 ; //1

    foreach($requisito->getCriterios() as $criterio) //2
    {
        $val1 = $criterio->getVarInterfaces()->getValoracion()->getValor() ; //3
        $ni1 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ; //3

        $influencia += $val1*$ni1 ; //3
        $pesos += $ni1 ; //3

        $val2 = $criterio->getVarDependencia()->getValoracion()->getValor() ; //3
        $ni2 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ; //3

        $influencia += $val2*$ni2 ; //3
        $pesos += $ni2 ; //3

        $val3 = $criterio->getVarDiferentesComportamientos()->getValoracion()->getValor() ; //3
        $ni3 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ; //3

        $influencia += $val3*$ni3 ; //3
        $pesos += $ni3 ; //3

        $val4 = $criterio->getVarFacilidadCambio()->getValoracion()->getValor() ; //3
        $ni4 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ; //3

        $influencia += $val4*$ni4 ; //3
        $pesos += $ni4 ; //3

        $val5 = $criterio->getVarRequisitosAsociados()->getValoracion()->getValor() ; //3
        $ni5 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ; //3
    }
}
```



```

    $influencia += $val5*$ni5 ;//3
    $pesos+= $ni5 ;//3

    $val6 = $criterio->getVarRestricciones()->getValoracion()->getValor() ;//3
    $ni6 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ;//3

    $influencia += $val6*$ni6 ;//3
    $pesos+= $ni6 ;//3

    $val7 = $criterio->getVarTransacciones()->getValoracion()->getValor() ;//3
    $ni7 = $criterio->getVarInterfaces()->getNivelImportancia()->getValor() ;//3

    $influencia += $val7*$ni7 ;//3
    $pesos+= $ni7 ;//3

} //4

// calculo la complejidad
$complejidad = 0 ;//5

if($pesos>0){ //6
    $complejidad = $influencia / $pesos ;//7
    $complejidad = round($complejidad) ;//7
} //8

return $complejidad ;//9
} //10

```

Figura 22 Código: calcularComplejidad (elaboración propia).

A continuación se crea el grafo de flujo asociado al algoritmo seleccionado, luego se realiza el cálculo de la complejidad ciclomática (V (G)) del mismo, ver Figura 23:

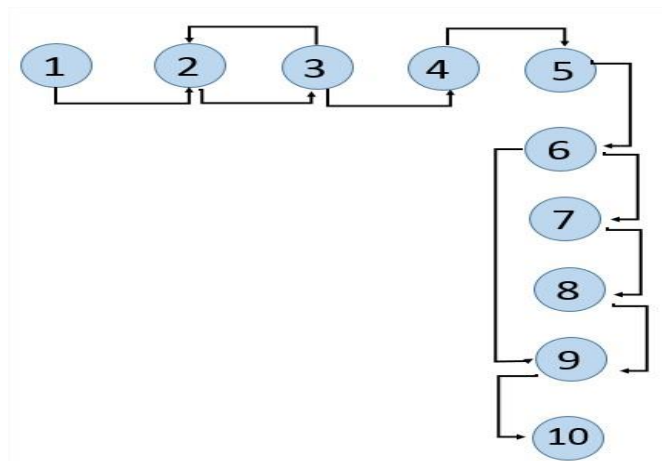


Figura 23 Grafo de flujo asociado al algoritmo calcularComplejidad (Requisito \$requisito) (elaboración propia).

Fórmulas para calcular la complejidad ciclomática:

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

Donde “A” es la cantidad de aristas y “N” la cantidad de nodos.

$$V(G) = (11 - 10) + 2$$

$$V(G) = 3$$

$$V(G) = P + 1$$

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

$$V(G) = 2 + 1$$

$$V(G) = 3$$

$$V(G) = R$$

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

$$V(G) = 3$$

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es tres. Esto significa que existen tres posibles caminos por donde el flujo puede circular y representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Número	Caminos básicos
1	1-2-3-4-5-6-7-8-9-10
2	1-2-3-4-5-6-9-10
3	1-2-3-2-3-4-5-6-7-8-9-10

Tabla 12 Caminos básicos del flujo (elaboración propia).

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: Calcular la complejidad a partir de un criterio experto.

Condición de ejecución: Que exista el menos un criterio establecido sobre el requisito.

Entrada:

- requisito: Insertar usuario.

Resultado esperado: Se espera que calcule la complejidad del requisito evaluado, actualizándose este valor en el listado de requisitos.

Resultado obtenido: Satisfactorio.

Caso de prueba para el camino básico # 2.

Descripción: Calcular la complejidad sin ningún criterio experto realizado aún.

Condición de ejecución: Que exista el menos un requisito registrado.

Entrada:

- requisito: Insertar usuario.

Resultado esperado: Se espera un mensaje indicando la ausencia de criterios realizados sobre el requisito.

Resultado obtenido: Satisfactorio.

Caso de prueba para el camino básico # 3.

Descripción: Calcular la complejidad a partir de varios criterios expertos.

Condición de ejecución: Que exista el menos un criterio establecido sobre el requisito.

Entrada:

- requisito: Insertar usuario.

Resultado esperado: Se espera que calcule la complejidad del requisito evaluado, actualizándose este valor en el listado de requisitos.

Resultado obtenido: Satisfactorio.

Una vez concluidos los casos de pruebas para cada camino básico, se puede constatar a partir de los resultados obtenidos, que todas las sentencias del algoritmo se ejecutan al menos una vez garantizando que el código es correcto.

Para detectar posibles funciones incorrectas, errores de interfaz o de estructuras de datos; fueron aplicadas pruebas de caja negra a la herramienta, específicamente la técnica de partición de equivalencias. Estas pruebas se llevaron a cabo en el departamento de Calidad del centro CEIGE.

Prueba de Caja negra

Para realizarlas no es necesario conocer los detalles internos del programa y su objetivo fundamental, es probar que el software está acorde a los requisitos. [53]

Para la ejecución de este tipo de pruebas se realizaron un conjunto de casos de prueba que tienen como objetivo principal determinar si los requisitos están parcial o completamente satisfactorios. A continuación se representa el diseño de caso de prueba para el RF: Importar requisitos.

Caso de prueba para el requisito Importar requisitos.

Condiciones de ejecución

- Se debe identificar y autenticar ante el sistema y además debe tener los permisos para ejecutar esta acción.
- Se debe seleccionar en el menú la opción Importar.

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Importar requisitos	El sistema permite importar los RF de software contenidos en el producto de trabajo Criterios para validar requisitos del producto.	EP 1.1: Importar requisitos entrando datos válidos.	<ul style="list-style-type: none">– Se introduce el número de la Hoja Excel donde están listados los requisitos.– Se introduce el número de la Columna donde están listados los requisitos.– Se introduce el número de la Fila Inicial donde comienza el listado de los requisitos.– Se introduce el número de la Fila Final donde termina el listado de los requisitos.– Se busca el archivo a Importar (Criterios para validar requisitos del producto.xls).– Se presiona la opción Aceptar
		EP 1.2: Importar requisitos dejando campos vacíos	<ul style="list-style-type: none">– Se busca el archivo a Importar (Criterios para validar requisitos del producto.xls).– Se presiona la opción Aceptar
		EP 1.3: Importar requisitos introduciendo datos incorrectos.	<ul style="list-style-type: none">– Se introducen datos inválidos en cualquiera de los campos de textos.– Se busca el archivo a Importar (Criterios para validar requisitos del producto.xls).

Tabla 13 Caso de prueba para el requisito Importar requisitos (elaboración propia).

Resultados de las pruebas

La herramienta fue sometida a dos iteraciones de pruebas de caja negra, encontrándose en una primera iteración, un total de tres no conformidades, distribuidas de la siguiente manera, ver figura 24:

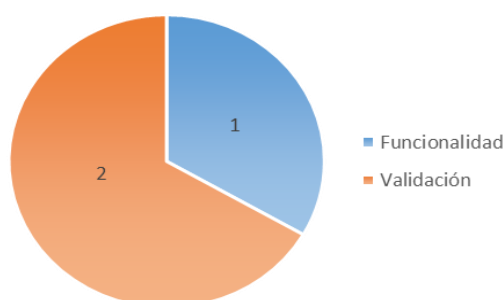


Figura 24 Resultados de las pruebas de caja negra (elaboración propia).

Todas estas no conformidades fueron resueltas, lo cual permitió que al realizar una 2da iteración no se detectaran no conformidades, arrojando resultados satisfactorios, que garantizaron una correcta utilización de la herramienta. A partir de esto se emitió el Acta de Liberación en la cual consta que la herramienta desarrollada cumple con las condiciones de calidad requerida.

Para lograr la aceptación de la herramienta por parte del cliente, se realizó su validación en términos de negocio para medir y comparar resultados.

Prueba de aceptación

En aras de validar la herramienta resultado de esta investigación, para demostrar el cumplimiento del objetivo propuesto de la misma, se decidió utilizarla en un proyecto real que ya tenía calculada la complejidad de los RF a través de la herramienta actual (Evaluación de requisitos.xls). El proyecto seleccionado fue Cedrux, perteneciente al centro CEIGE. Por ser Cedrux un proyecto muy grande, el cálculo de la complejidad de los RF se hizo desde la primera vez por subsistemas. Decidiéndose por la importancia y cantidad de requisitos que contiene volverle a calcular la complejidad a los del subsistema Contabilidad, núcleo de Cedrux.

Cálculo de la complejidad de los RF de software del proyecto Cedrux: subsistema Contabilidad utilizando la herramienta definida en la UCI actualmente

El subsistema Contabilidad tiene un total de 107 RF de software, con la ayuda de Tamara Rodríguez (analista principal del proyecto) se pudo consultar el producto de trabajo Evaluación de requisitos. En el mismo se comprobó que se había calculado la complejidad de los 107 RF, arrojando el siguiente resultado:

Complejidad	Cantidad de requisitos
Alta	27
Media	25
Baja	55

Tabla 14 Resultados Evaluación de requisitos.xls (elaboración propia).

Cálculo de la complejidad de los RF de software del proyecto Cedrux: subsistema Contabilidad utilizando la herramienta resultado de la investigación

A partir de los parámetros que define el nuevo método para calcular la complejidad de los RF, se insertaron los 107 requisitos en la herramienta para su posterior cálculo de la complejidad. Estos fueron evaluados por Noidis Barroso (analista) y Yarenis González (desarrolladora), de esta forma se cumple que múltiples expertos den criterios sobre los RF. Una vez evaluados el resultado obtenido es el siguiente:

Complejidad	Cantidad de requisitos
Alta	38
Media	40
Muy alta	2
Muy baja	0
Baja	27

Tabla 15 Resultados COMREQ (elaboración propia).

Análisis de los resultados obtenidos

Los resultados obtenidos fueron presentados a un grupo de especialistas que formaron parte del proyecto Cedrux. Los cuales en su calidad de expertos en el tema se mostraron satisfechos con el cálculo de la complejidad de los RF arrojado por la nueva herramienta, dado que muestran una mayor proximidad de la complejidad real de los requisitos evaluados. A partir de esto se acordó emitir el Acta de aceptación en la cual consta que la herramienta desarrollada minimiza las deficiencias de la herramienta actual y satisface las necesidades de los usuarios, ver Anexo 4.

Conocer el estado de satisfacción del usuario respecto a la herramienta propuesta es de gran utilidad para la validación de la presente investigación. A continuación se describe la aplicación de la técnica ladov.

3.6 Validación de satisfacción del usuario. Aplicación de la Técnica de ladov.

ladov es una técnica que permite el estudio del grado de satisfacción de los involucrados en un proceso o actividad objeto de análisis. Esta técnica ha sido ampliamente utilizada por su carácter

genérico [54]. La técnica está conformada por cinco preguntas: tres cerradas y dos abiertas las cuales son reformuladas en la presente investigación para evaluar la satisfacción de los usuarios sobre la herramienta. A partir de las preguntas se conforma el “Cuadro Lógico de ladov” que establece la relación entre las preguntas cerradas, indicando la posición de cada persona en la escala de satisfacción.

¿Las salidas de la herramienta, complejidad del requisito satisface sus necesidades relacionadas con este tema?	¿Considera usted que se debe continuar realizando la determinación de la complejidad de los requisitos funcionales por la herramienta anterior?								
	No			No sé			Sí		
	¿Utilizaría usted la herramienta propuesta para la determinación de la complejidad de los requisitos funcionales en los proyectos de desarrollo de software?								
	Si	No Sé	No	Si	No Sé	No	Si	No Sé	No
Me gusta mucho	1	2	6	2	2	6	6	6	6
No me gusta mucho	2	2	3	2	3	3	6	3	6
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	5
No sé qué decir	2	3	6	3	3	3	6	3	4

Tabla 16 Cuadro Lógico de ladov. (Modificado por el autor).

El número resultante de la interrelación de las tres preguntas indica la posición de cada cual en la escala de satisfacción siguiente [54]:

1. Clara satisfacción
2. Más satisfecho que insatisfecho
3. No definida
4. Más insatisfecho que satisfecho
5. Clara insatisfacción
6. Contradictoria

La información relacionada con 107 RF correspondientes al subsistema contabilidad del proyecto CEDRUX fue mostrada a un grupo de usuarios.

Para medir el grado de satisfacción se tomó una muestra de 9 usuarios, teniendo en cuenta los años de experiencia en la producción y roles afines al área de Administración de requisitos. A partir de esto se aplicó la técnica de ladov para medir el nivel de satisfacción respecto a los

resultados arrojados por la herramienta. El resultado de la evaluación de la satisfacción individual fue el siguiente, según la escala de satisfacción:

Nivel de satisfacción	Cantidad
Clara satisfacción	7
Más satisfecho que insatisfecho	2

Tabla 17 Resultados de la aplicación de la técnica de ladov (elaboración propia).

Para obtener el índice de satisfacción grupal (ISG) se procesan los criterios de las personas de acuerdo a los niveles de satisfacción que se expresan en la escala numérica que oscila entre +1 y - 1 de la siguiente forma [54]:

- + 1 Máximo de satisfacción
- 0,5 Más satisfecho que insatisfecho
- 0 No definido y contradictorio
- - 0,5 Más insatisfecho que satisfecho
- - 1 Máxima insatisfacción

La ISG se calcula por la siguiente fórmula:

$$ISG = \frac{A(+1) + B(+0,5) + C(0) + D(-0,5) + E(-1)}{N} \quad (1)$$

En esta fórmula *A*, *B*, *C*, *D*, *E*, representan el número de sujetos con índice individual 1; 2; 3 ó 6; 4; 5 y donde *N* representa el número total de sujetos del grupo.

Los valores que se encuentran comprendidos entre - 1 y - 0,5 indican insatisfacción; los comprendidos entre - 0,49 y + 0,49 evidencian contradicción y los que están entre 0,5 y 1 indican que existe satisfacción.

En este caso ISG fue de 0,88 lo que representa un alto grado de satisfacción de los usuarios al haber utilizado la herramienta propuesta.

El ladov contempla además dos preguntas complementarias de carácter abierto.

1. ¿Considera útil la posibilidad de determinar la complejidad de los requisitos a partir de importar el documento criterios para validar requisitos del producto?
2. ¿Qué elemento usted adicionaría a la herramienta que se propone?

La principal recomendación de los usuarios radicó en la posibilidad de introducir nuevas variables que definan la complejidad del requisito a partir del análisis que se haga del mismo. Se considera de muy útil la posibilidad de valorar los diferentes criterios de expertos, así como la existencia de una herramienta que permita la unificación de estos criterios, permitiendo determinar la complejidad de los RF de software con un mayor grado de cercanía al juicio humano.

La aplicación de la técnica de ladov arrojó resultados satisfactorios que validan la propuesta realizada de acuerdo a la satisfacción de los usuarios, además de que fueron considerados los

criterios expresados para futuras mejoras a la herramienta. Para consultar la encuesta ver Anexo 5.

3.7 Conclusiones parciales

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

- Para definir la arquitectura de la herramienta previa a su implementación, fue creado el diagrama de componentes identificando las dependencias lógicas entre los componentes de software que intervienen en el desarrollo de la misma.
- La definición de los estándares de codificación a utilizar durante la implementación de la herramienta, permitieron un mejor entendimiento del código, así como la reutilización de este para la realización de posibles modificaciones futuras.
- El desarrollo del diagrama de despliegue, permitió conocer la relación entre la arquitectura de software y la arquitectura de hardware de la herramienta.
- Fueron aplicadas las métricas TOC y RC para validar el diseño de la herramienta, lo que permitió comprobar que sus respectivos atributos de calidad se comportan de forma favorable para lograr un bajo acoplamiento, una alta reutilización e implementación sencilla.
- Para examinar la lógica interna de la estructura de la herramienta desarrollada, fue aplicada la técnica camino básico, la cual aseguró que se ejecutara al menos una vez cada sentencia del algoritmo seleccionado.
- La herramienta fue liberada en dos iteraciones de prueba de caja negra, definidas en una primera, tres no conformidades (ninguna significativa) y en una segunda iteración todas estas fueron resueltas garantizando así la calidad de la misma.
- Se utilizó la herramienta desarrollada en el proyecto Cedrux para realizar el cálculo de la complejidad de los 107 RF del subsistema Contabilidad (núcleo de Cedrux); los resultados obtenidos mostraron una mayor proximidad a la complejidad real una vez terminado este subsistema.
- Se aplicó la técnica ladov, lo que permitió conocer el grado de satisfacción de los usuarios.

Conclusiones generales

La culminación del presente trabajo de diploma permite arribar a las siguientes conclusiones:

- Con el estudio de diferentes herramientas que gestionan requisitos se pudo concluir que solo la implementada en Excel realiza el cálculo de la complejidad de los RF de software y que ninguna hace uso de técnicas de soft computing para realizarlo.
- La complejidad de los RF es una información cualitativa que no puede ser evaluada de forma precisa, siendo necesario el tratamiento de la incertidumbre en la misma con la utilización de técnicas de soft computing, propuestas en el Método para determinar la complejidad de los RF de software.
- Las herramientas y tecnologías seleccionadas permitieron realizar el análisis, diseño, implementación y validación de la solución teniendo en cuenta la plataforma de trabajo de la UCI y en correspondencia con el principio de independencia tecnológica en el cual está basado actualmente el desarrollo de software en Cuba.
- A través de entrevistas y tormenta de ideas se capturaron los RF necesarios (18) para la implementación de la herramienta, estos fueron validados con los prototipos de interfaz de usuario.
- La generación de los artefactos correspondientes a las disciplinas de la metodología usada para el desarrollo de la presente investigación, permitieron guiar el proceso de implementación de la herramienta resultado de la misma.
- Las pruebas de software y la aplicación de la técnica Iadov, permitieron validar la herramienta en términos de calidad y el grado de satisfacción del usuario respectivamente.

Recomendaciones

Se recomienda para la continuidad de la investigación:

- Desarrollar un módulo de administración para la gestión de las variables y clasificaciones que permiten calcular la complejidad de los RF.
- Incorporar al área de proceso de Administración de requisitos la herramienta definida en la presente investigación, permitiendo la extensión de esta a todos los centros de desarrollo de la UCI.

Referencias Bibliográficas

1. Manifesto, C., *Think Big, Act Small*. The Standish Group International Inc, 2013.
2. Pressman, R.S., *Ingeniería del Software: Un enfoque práctico*. 1997: Mikel Angoar.
3. Mustelier, D., *Método para determinar la complejidad de los requisitos funcionales de software*. 2014, Universidad de las Ciencias Informáticas: La Habana.
4. Martinto, M.P.C., *El diseño metodológico de la investigación científica. Teoría de Muestreo: población y muestra. Diseño experimental y métodos*. 2011.
5. Sommerville, I., *Ingeniería del Software*. 2005. **7**.
6. Boehm, B.W., *Software Engineering*. IEEE Trans. Comput., 1976. **25**(12): p. 1226-1241.
7. Chaves, M.A., *La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software*. 2006.
8. Lynch, J., *New Standish Group report shows more project failing and less successful projects*. Press release. Standish Group, Boston, MA, Retrieved, 2009. **21**.
9. Dominguez, J., *The Curious Case of the CHAOS Report 2009 [online],[cited 1-06-2011]*. Available from Internet: <http://www.projectsart.co.uk/the-curious-case-of-the-chaos-report-2009.html>, 2009.
10. September, A., *IEEE standard glossary of software engineering terminology*. Office, 1990. **121990**(1): p. 1.
11. Chaves, M.A., *La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software*. InterSedes, 2011. **6**(10).
12. Méndez, G., *Ingeniería de Requisitos*.
13. Escalona, M.J. and N. Koch, *Ingeniería de Requisitos en Aplicaciones para la Web—Un estudio comparativo*. Universidad de Sevilla, 2002.
14. EcuRed, *Validación de Requisitos*.
15. Zadeh, L.A., *Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems*. Soft Computing-A fusion of foundations, methodologies and applications, 1998. **2**(1): p. 23-25.
16. IBM, *Rational RequisitePro tutorials*. 2012.
17. IBM, *IBM Rational DOORS getting started*. 2013.
18. IPCorp, *OSRMT: Open Source Requirements Management Tool*. 2008.
19. GatherSpace, *Cloud Requirements Management Solution*. 2014.
20. Callejas, M., L. Castillo, and R. Fernández, *Heler: Una Herramienta para la ingeniería de requisitos automatizada*. 2010. **6**.
21. Rodríguez, T., *Metodología de desarrollo para la actividad productiva de la UCI*. 2014.
22. Larman, C., *UML y Patronos*. 1999: Pearson.
23. Paradigm, V., *Visual paradigm for uml*. Visual Paradigm for UML-UML tool for software application development, 2010.
24. IBM, *Rational Rose Enterprise*. 2013.

25. Gil Escudero, F.A., *Sistema informático para el manejo de personal de la Dirección de Impuestos y Aduanas Nacionales Seccional Pereira*. 2011.
26. Sánchez, J., *Introducción a PHP*. 2012.
27. Belmonte, O., *Introducción al lenguaje de programación Java. Una guía básica*. 2005.
28. Gutiérrez, J.J., *¿ Qué es un framework web?* 2006.
29. Frederick, S. and C.N. Ramsay, *Learning Ext JS: Build Dynamic, Desktop-style User Interfaces for Your Data-driven Web Applications*. 2008: Packt Publishing Ltd.
30. symfony, *Symfony para programadores*. 2015.
31. Boudreau, T., et al., *NetBeans: the definitive guide*. 2002: " O'Reilly Media, Inc."
32. Gajda, W., *Instant PhpStorm Starter*. 2013: Packt Publishing Ltd.
33. Rivera, C. and H. Martin, *SISTEMAS GESTORES DE BASE DE DATOS (SGBD)*. 2012.
34. MySQL, A., *MySQL*. 2001.
35. Momjian, B., *PostgreSQL: introduction and concepts*. Vol. 192. 2001: Addison-Wesley New York.
36. Bourdon, R., *WampServer, a Windows web development environment*. Retrieved June, 2013. **25**: p. 2013.
37. Castilla-LaMancha, U.d., *CURSO DE FORMACIÓN INTERNET*. 2008.
38. Davis, S.J. and K.M. Murphy, *A competitive perspective on internet explorer*. American Economic Review, 2000: p. 184-187.
39. Buschmann, F., et al., *Pattern-oriented software architecture: a system of patterns*. 1996. Part II, 2001.
40. González, Y.D. and Y.F. Romero, *Patrón Modelo-Vista-Controlador*. Revista Telem@ tica, 2012. **11**(1): p. 47-57.
41. Rojas, M.J.C.O., *Patrones de Diseño*.
42. Gracia, J., *Patrones de diseño*. 2005.
43. *Gestión de la Calidad y Pruebas de Software*.
44. Studio, V., *Diagramas de secuencia UML: Referencia*. MSDN.
45. Cabrera, L.V., *Bases de Datos Orientadas a Objetos*.
46. Tedeschi, N., *¿Qué es un Patrón de Diseño?* MSDN, 2013.
47. Hamon, H., *Applying Design Patterns to Symfony*. 2014.
48. Nina, D.M., D. García Quispe, and M.M. Sotes Carrillo, *Artefacto: Diagrama de componentes*. 2009.
49. Guglielmetti, M., et al., *Master Magazine*. 2005.
50. Herranz, R., *Utópica Informática*. 2010.
51. Jacobson, I., G. Booch, and J. Rumbaugh, *El proceso unificado de desarrollo de software*. Vol. 7. 2000: Addison Wesley Reading.
52. Usaola, M.P., *Pruebas del Software*.

53. López, C., R. Marticorena, and D.H. Martín, *Pruebas de caja negra: una experiencia real en laboratorio*. Actas de las XI Jornadas de Enseñanza Universitaria de Informática, Jenui 2005, 2005: p. 189-196.
54. López, A. and V. González, *La técnica de ladov. Una aplicación para el estudio de la satisfacción de los alumnos por las clases de Educación Física*. Revista Digital Lecturas: Educación Física y Deporte, 2002. **8**(47).

Anexos

Anexo 1: Metodología AUP-UCI

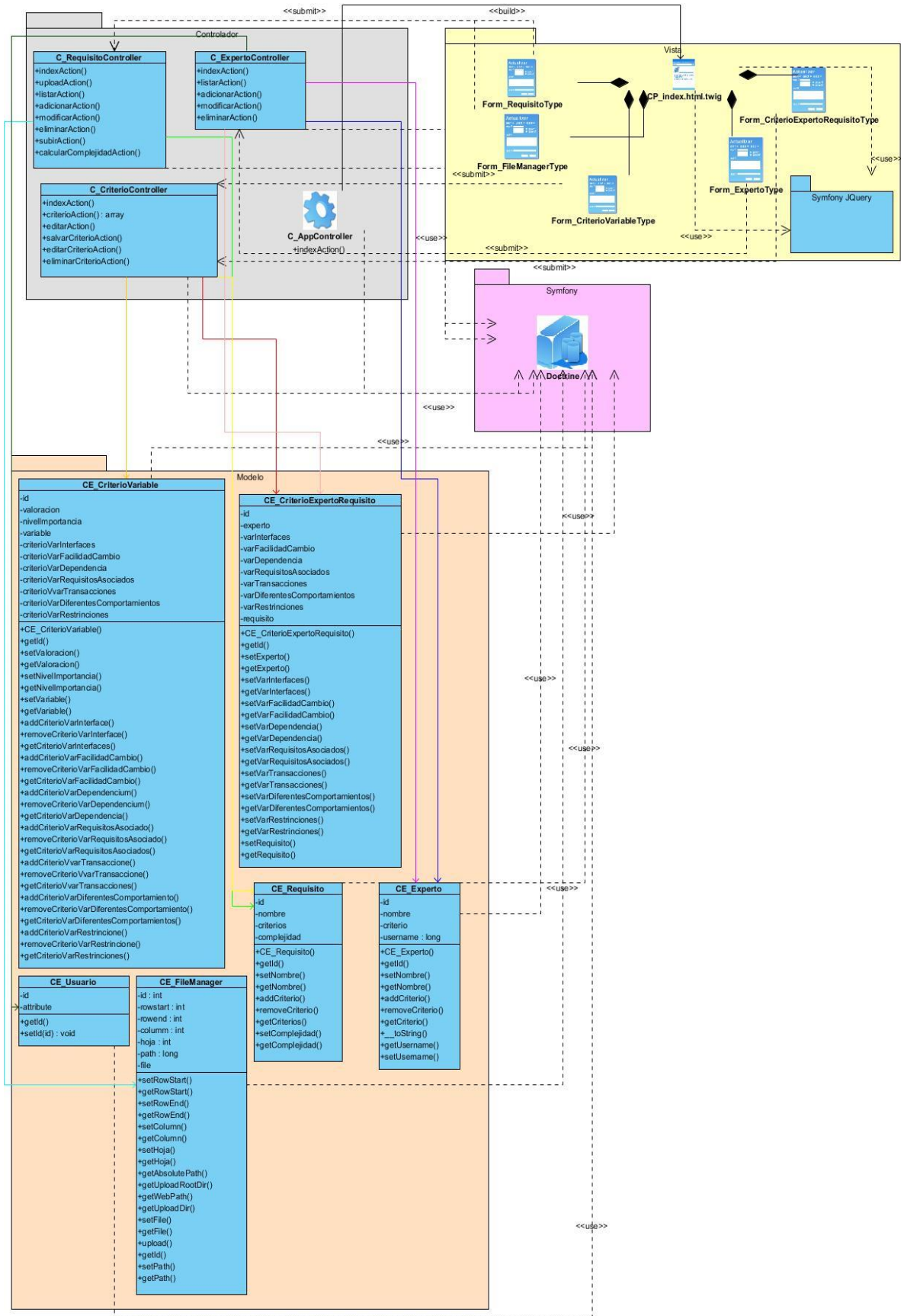
Fases AUP	Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Además, en la transición se capacita a los usuarios finales sobre la utilización del software.
Construcción		
Transición		
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Anexo 2: GRASP


Patrones de Software Generales para la Asignación de Responsabilidades (GRASP)

<i>Patrón</i>	<i>Descripción</i>
Experto en Información	<p>¿Un principio general del diseño de objetos y la asignación de responsabilidades?</p> <p>Asigne una responsabilidad al experto en información, —la clase que tiene la información necesaria para llevar a cabo la responsabilidad.</p>
Creador	<p>¿Quién crea? (Nótese que la Factoría es una solución alternativa frecuente.)</p> <p>Asigne a la clase B la responsabilidad de crear una instancia de la clase A si se cumple alguno de los puntos siguientes:</p> <ol style="list-style-type: none"> 1. B contiene a A 2. B agrega a A 3. B tiene los datos de inicialización de A 4. B registra a A 5. B utiliza estrechamente a A
Controlador	<p>¿Quién gestiona un evento del sistema?</p> <p>Asigne la responsabilidad de gestionar un mensaje de un evento del sistema a una clase que represente una de estas opciones:</p> <ol style="list-style-type: none"> 1. Representa el sistema global, dispositivo o un subsistema (controlador de fachada). 2. Representa un escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de caso de uso o sesión).
Bajo Acoplamiento (evaluativo)	<p>¿Cómo dar soporte a las bajas dependencias y al incremento de la reutilización?</p> <p>Asigne responsabilidades de manera que el acoplamiento (innecesario) se mantenga bajo.</p>
Alta Cohesión (evaluativo)	<p>¿Cómo mantener manejable la complejidad?</p> <p>Asigne responsabilidades de manera que la cohesión permanezca alta.</p>
Polimorfismo	<p>¿Quién es el responsable cuando el comportamiento varía en función del tipo?</p> <p>Cuando las alternativas o comportamientos relacionados varían según el tipo (clase), asigne la responsabilidad del comportamiento —utilizando operaciones polimórficas— a los tipos para los que varía el comportamiento.</p>
Fabricación Pura	<p>¿Quién es el responsable cuando está desesperado, y no quiere violar los principios de alta cohesión y bajo acoplamiento?</p> <p>Asigne un conjunto altamente cohesivo de responsabilidades a una clase de "comportamiento" artificial o de conveniencia que no representa un concepto del dominio del problema —algo inventado—, para dar soporte a la alta cohesión, bajo acoplamiento y la reutilización.</p>
Indirección	<p>¿Cómo asignar responsabilidades para evitar el acoplamiento directo?</p> <p>Asigne la responsabilidad a un objeto intermedio para mediar entre otros componentes o servicios, de manera que no se acoplan directamente.</p>
Variaciones Protegidas	<p>¿Cómo asignar responsabilidades a los objetos, subsistemas, y sistemas de manera que las variaciones o inestabilidad en estos elementos no influya de manera no deseable en otros elementos?</p> <p>Identifique los puntos de variaciones predecibles o inestabilidad; asigne las responsabilidades para crear una "interfaz" estable alrededor de ellos.</p>

Anexo 3: Diagrama de clases del diseño

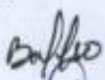





Anexo 4: Acta de aceptación


CENTRO DE INFORMATIZACIÓN DE GESTIÓN DE ENTIDADES

Producto: COMREQ.


Firma de los involucrados en el proceso:


- Por la parte del Cliente:
 - Noidis Barroso Hidalgo 
 - William González Obregón 
 - Yareni Echeandía González 
- Especialista del proyecto (CONREQ):
 - Alejandro Alberto Ramírez Toranzo 

Observaciones del proceso:
Teniendo en cuenta que las No Conformidades han sido debidamente resueltas y validada la eficacia de la corrección por parte del cliente, se ha tomado el acuerdo de Aceptar con fecha 28 de Mayo de 2015, la Herramienta para la determinación de la complejidad de los requisitos de software.

Para que conste la Aceptación, dando fe del acuerdo, se extiende la presente Acta en dos (2) ejemplares, rubricados por los principales Representantes de las Partes.

Entrega [Parte Suministradora] Recibe [Representante Parte Cliente]

Nombre y Apellidos: Alejandro Alberto Ramírez Toranzo Nombre y Apellidos: William González Obregón
Cargo: Estudiante Cargo: Especialista
Firma: 



Referencia:

Anexo 5: Encuesta

Encuesta para conocer el grado de satisfacción de los usuarios con relación a la aplicación de la herramienta para la determinación de la complejidad de los requisitos funcionales de software utilizando técnicas de soft computing.

Datos de interés

Centro: *CEZGE*

Cargo (si está ocupando alguno): *Analista (Especialista)*

Proyecto: *Cedruz*

Rol: *Analista*

Preguntas:

¿Considera usted que se debe continuar realizando la determinación de la complejidad de los requisitos funcionales por la herramienta anterior?

Si No No sé

¿Utilizaría usted la herramienta propuesta para la determinación de la complejidad de los requisitos funcionales en los proyectos de desarrollo de software?

Si No No sé

¿Las salidas de la herramienta, complejidad del requisito satisface sus necesidades relacionadas con este tema?

Me gusta mucho

No me gusta mucho

Me da lo mismo

Me disgusta más de lo que me gusta

No me gusta nada

No sé qué decir

¿Considera útil la posibilidad de determinar la complejidad de los requisitos a partir de importar el documento criterios para validar requisitos del producto?

Si No No sé

¿Qué elemento usted añadiría a la herramienta que se propone?
