

Universidad de las Ciencias Informáticas



# **Herramienta para la identificación de requisitos no funcionales de software**

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

**Autor: José Ernesto Lenzano Ramírez**

## **Tutores:**

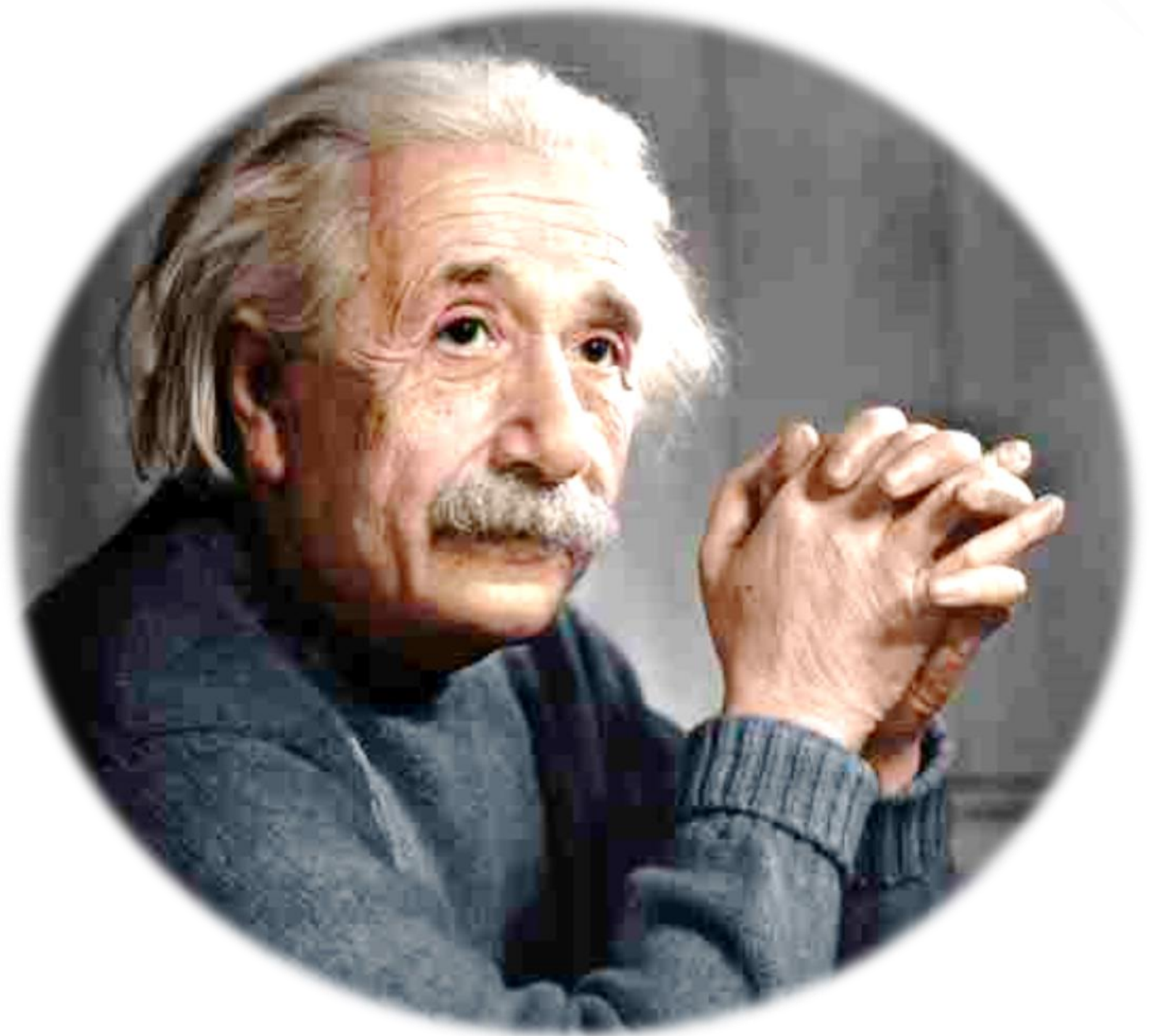
Ms. C. Yenisel Molina Hernández

Ing. Madelin Haro Pérez

Dr. C. Arturo Orellana García

La Habana, 2019

“Año 61 de la Revolución”



*"La medida de la inteligencia es la capacidad de cambiar."*

*Albert Einstein*

## Agradecimientos

Quiero agradecer a todas las personas que han hecho posible este sueño.

- **En primer lugar, a mi familia, en especial a mi mamá, a mi papá, mis hermanos y mi novia**, por siempre de una manera u otra estar apoyándome en cada paso que doy.
- **A mis abuelas, mis tías, mi prima, mi padrastro**, en especial a mi abuelita Martha que siempre estarán ahí para mí.
- **A mis amigos y hermanos** que han estado durante toda la carrera y me ha tocado conocer, reír y llorar donde quiere que esté nunca los olvidaré.
- **A mis tutores** por haber confiado en mí, por todo lo que aprendí con ustedes y por toda la ayuda brindada.
- **A todos** los que se me quedan y que saben que estuvieron ahí, muchas gracias.

## Declaración de autoría

Declaro ser el único autor de la tesis "Herramienta para la identificación de requisitos no funcionales". Reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación con carácter exclusivo. Para que así conste firmo, junto a mis tutores, la presente declaración a los \_\_\_\_ días del mes de junio del año 2019.

---

Firma del Autor

Jose Ernesto Lenzano Ramirez

---

Firma del Tutor

Ms. C. Yenisel Molina Hernández

---

Firma del Tutor

Ing. Madelín Haro Pérez

---

Firma del Tutor

Dr. C. Arturo Orellana García

## Resumen

Los requisitos, elemento esencial en el desarrollo de software, identifican las necesidades funcionales y de comportamiento de un sistema, según el cliente. Su elicitación es un proceso que requiere un conjunto de habilidades generales y profesionales que se ganan con la experiencia profesional y con la participación en numerosas elicitaciones. Su complejidad es reconocida en el mundo de la informática siendo más marcada para la identificación de los requisitos no funcionales puesto que el usuario no está seguro de cómo quiere que se vea el sistema, qué rendimiento debe tener o cuáles son los niveles de seguridad o qué hardware y/o software son necesarios.

Teniendo en cuenta que la Universidad de las Ciencias Informáticas trabaja la producción de software para un mercado diverso y en aumento y que sus equipos de proyecto no siempre tienen la experticia suficiente, este trabajo tiene el objetivo de desarrollar una herramienta que apoye el trabajo de los analistas de sistemas en la identificación de requisitos no funcionales. Se emplea la técnica de reutilización de requisitos estableciendo como base los proyectos ya realizados en los centros de desarrollo de la universidad. Se obtiene una herramienta computacional basada en las tecnologías libres Java 8.0.91 y PostgreSQL 9.3 que, dado el tipo, ámbito, área del proyecto, subcaracterística y métrica ofrece como resultado un conjunto de requisitos no funcionales que pueden asociarse a él. Además, se obtiene el marco teórico que fundamenta el desarrollo de la herramienta y la documentación técnica, útil para procesos posteriores de mantenimiento.

**Palabras clave:** elicitación de requisito, ingeniería de requisito, requisito no funcional, reutilización de requisitos.

## **Abstract**

The requirements, an essential element in software development, identify the functional and behavioural needs of a system, according to the customer. Its elicitation is a process that requires a set of general and professional skills that are gained through professional experience and participation in numerous elicitations. Its complexity is recognized in the world of information technology being more marked for the identification of non-functional requirements since the user is not sure how he wants the system to look, what performance it should have or what are the security levels or what hardware and / or software are needed.

Bearing in mind that the University of Computer Science is working on software production for a diverse and growing market and that its project teams do not always have sufficient expertise, this work aims to develop a tool that supports the work of systems analysts in identifying non-functional requirements. The technique of reuse of requirements is used, establishing as a base the projects already carried out in the development centers of the university. A computational tool is obtained based on the free technologies Java 8.0.91 and PostgreSQL 9.3 which, given the type, scope, project area, subfeature and metric, offers as a result a set of non-functional requirements that can be associated with it. In addition, the theoretical framework is obtained that bases the development of the tool and the technical documentation, useful for later processes of maintenance.

**Keywords:** non-functional requirement, requirement elicitation, requirement engineering, reuse of requirements.

# Índice

Introducción .....	1
Capítulo 1. Fundamentación teórica sobre la identificación de requisitos no funcionales y su reutilización ..	6
1.1    La elicitación de requisitos de software.....	7
1.1.1    Requisitos no funcionales de software.....	8
1.1.2    Técnica de reutilización .....	10
1.2    Herramientas informáticas relacionadas a la gestión e identificación de los requisitos .....	12
1.3    Metodologías de desarrollo.....	14
1.4    Tecnologías y herramientas.....	16
1.4.1    Lenguaje de programación .....	16
1.4.2    Tecnologías y herramientas.....	16
1.4.3    Herramienta CASE y lenguaje de modelado.....	18
1.4.4    Sistema de gestión de base de datos .....	18
1.5    Conclusiones del capítulo .....	19
Capítulo 2. Descripción de la herramienta para apoyar la elicitación de requisitos no funcionales .....	20
2.1    Modelado del negocio.....	21
2.2    Identificación de requisitos.....	23
2.2.1    Casos de uso del sistema.....	24
2.3    Análisis y diseño .....	33
2.3.1    Modelo de datos .....	37
2.3.2    Patrones de diseño utilizados .....	37
2.4    Conclusiones del capítulo .....	41
Capítulo 3. Implementación y pruebas de la herramienta para la identificación de requisitos no funcionales de software .....	42
3.1    Implementación .....	42
3.1.1    Despliegue de la herramienta .....	47

3.2 Pruebas internas.....	48
3.3 Pruebas de aceptación .....	57
3.4 Conclusiones del capítulo .....	59
Conclusiones .....	60
Recomendaciones .....	61
Referencias bibliográficas .....	62
Anexos.....	65
Anexo 1: Gestionar proyecto.....	65
Anexo 2: Gestionar métricas .....	65
Anexo 3: Gestionar métricas .....	65
Anexo 4: Gestionar usuario.....	65
Anexo 5: Gestionar centros de producción.....	66
Anexo 6: Gestionar tipos de proyectos.....	66
Anexo 7: Gestionar ámbito de proyectos.....	66
Anexo 8: Descripción de las variables de los casos de usos.....	66



## Índice de figuras

Figura 1-1. Relación de las actividades de la ingeniería de requisitos (Sommerville 2015) .....	7
Figura 1-2. Tipos de requisitos no funcionales. (Sommerville 2015).....	9
Figura 1-3. Modelo de calidad del producto de la ISO 25010 (Oficina Nacional de Normalización 2016) ...	10
Figura 2-1. Representación gráfica del método de reutilización de requisitos SIREN. Fuente: Elaboración propia.....	21
Figura 2-2. Modelo conceptual del negocio. Fuente: Elaboración propia .....	22
Figura 2-3 Casos de uso del sistema. Fuente: Elaboración propia.....	26
Figura 2-4: Arquitectura MVC. Fuente: Elaboración propia .....	33
Figura 2-5 Ejemplo de la comunicación entre las capas del patrón MVC en la solución. Fuente: Elaboración propia .....	34
Figura 2-6. Diagrama de paquetes. Fuente: Elaboración propia .....	35
Figura 2-7. Diagrama de clases de la solución. Fuente: Elaboración propia .....	36
Figura 2-8. Modelo de datos de la solución. Fuente: Elaboración propia .....	37
Figura 2-9. Ejemplo del uso del patrón controlador utilizando <i>IFlow</i> . Fuente: Elaboración propia .....	38
Figura 2-10. Ejemplo del uso del patrón creador utilizando <i>Flow</i> . Fuente: Elaboración propia .....	39
Figura 2-11. Ejemplo del uso del patrón alta cohesión. Fuente: Elaboración propia.....	40
Figura 2-12. Ejemplo del uso del patrón experto. Fuente: Elaboración propia.....	41
Figura 3-1. Ejemplo visualizar la aplicación de estándar de código para identificadores. Fuente: Elaboración propia .....	43
Figura 3-2. Ejemplo visualizar la aplicación de estándar de código para espacio en expresiones y sentencias. Fuente: Elaboración propia .....	44
Figura 3-3. Fragmento de código que representa el uso del estándar de codificación Importaciones. Fuente: Elaboración propia .....	45
Figura 3-4. Fragmento de código que muestra el uso de comentarios. Fuente: Elaboración propia.....	46
Figura 3-5 Fragmento de código que muestra el uso de variables y constantes. Fuente: Elaboración propia .....	46

Figura 3-6. Fragmento de código que muestra el uso de alineación y espacios en blanco. Fuente: Elaboración propia .....	47
Figura 3-7. Diagrama de despliegue. Fuente: Elaboración propia .....	48
Figura 3-8. Resultados de la prueba realizada en JUnit. Fuente: Elaboración propia.....	50
Figura 3-9. No conformidades encontradas en cada iteración de la técnica partición de equivalencia. Fuente: Elaboración propia .....	57
Figura 3-10. Acta de aceptación emitida por el Jefe del proyecto XAVIA-SIDEC .....	59

## Índice de tablas

Tabla 2-1 Actores del sistema. Fuente: Elaboración propia .....	25
Tabla 2-2 Descripción del CU 1. Gestionar el repositorio de requisitos. Fuente: Elaboración propia.....	26
Tabla 2-3 Descripción del CU 10 Identificar requisitos no funcionales. Fuente: Elaboración propia .....	30
Tabla 2-4 Descripción del CU11 Reutilizar requisitos no funcionales. Fuente: Elaboración propia.....	31
Tabla 3-1. Caso de prueba de caja negra “Gestionar repositorio de requisitos”. Fuente: Elaboración propia .....	51
Tabla 3-2. Caso de prueba de caja negra “Identificar RNF”. Fuente: Elaboración propia.....	55

## Introducción

Reportes de estudios de tendencias en la producción de sistemas informáticos como el proporcionado por la consultora internacional (Standish Group 2019), aseveran que una de las principales causas de falla de los proyectos es la incorrecta aplicación de la Ingeniería de Software en la definición, especificación, y/o administración de los requisitos. Datos y experiencias recopiladas por investigadores del tema como (Sommerville 2015) y (Pressman 2010) concuerdan que la probabilidad de que un software alcance el éxito aumenta en la medida en que se satisfagan las necesidades del usuario.

La ingeniería de requisitos (IR) es el proceso de descubrir, analizar, documentar y verificar los requisitos para un sistema. Por su parte, un requisito es una descripción de una condición o capacidad que debe cumplir un sistema, ya sea derivada de una necesidad del usuario identificada, o bien, estipulada en un contrato, estándar, especificación u otro documento formalmente impuesto al inicio del proceso (Sommerville 2015). Reflejan la necesidad de resolver problemas tanto del funcionamiento del sistema como de su comportamiento.

Existen varias clasificaciones de los requisitos de acuerdo al modo en que se analice las necesidades del sistema. Si el punto de mira radica en qué debe hacer, el requisito se considera funcional; si lo que define son las características de comportamiento del sistema como fiabilidad, tiempo de respuesta, capacidad de almacenamiento, seguridad, disponibilidad; entonces se clasifica como no funcional. A su vez los requisitos no funcionales se clasifican en: requisitos del producto, requisitos organizacionales y requisitos externos (Sommerville 2015). Una etapa de crucial importancia dentro de la RE es la obtención de los requisitos, conocida en el lenguaje de la profesión como elicitación, pero también tratada como levantamiento, captura o identificación de requisitos.

La elicitación de requisitos no es una actividad fácil de lograr. Los requisitos no son obvios y pueden provenir de muchas fuentes: clientes, usuarios, administradores, documentos, reglamentaciones, entre otros. Frecuentemente son difíciles de expresar en palabras, fundamentalmente porque el usuario no sabe explicar lo que hace de forma rutinaria o expresa lo que se hace mal sin llegar a concretar en ninguna necesidad. Los clientes no tienen claras sus necesidades y expectativas sobre el software, incluso la información obtenida por el ingeniero puede variar según la persona a la que consulte o las preguntas que se usen. De ahí que sean más fáciles de obtener los requisitos excepcionales antes que los comunes o rutinarios. (Gómez 2012). Para el caso de los requisitos no funcionales, (Pressman 2010) considera que es importante prestar especial atención por cómo puede condicionar el éxito en el desarrollo de las aplicaciones de software. Su afirmación se basa en que son críticos para el software; los clientes pueden encontrar la forma de trabajar alrededor de una funcionalidad que no cubra todas sus necesidades, pero el

incumplimiento de un requisito no funcional puede llevar al sistema a ser inutilizable por causas tan simples como ser inseguro, no usable, o no se ejecute sobre la infraestructura existente.

En ocasiones los requisitos hacen referencia a normativas, leyes, regulaciones y estándares que el proyecto debe cumplir y en otros casos, son familias de productos que comparten un núcleo de funcionalidades comunes o tal vez funcionalidades completas de un sistema o aplicación que se desean emplear en otros. (Visure 2018) La experiencia que acumule el grupo de profesionales que trabaje la elicitación de requisitos, normalmente se sustenta en un número considerable de veces que ha ejecutado esta acción. Los proyectos con características comunes como, por ejemplo: desarrollo en una misma plataforma, semejanzas en las peticiones de las interfaces, responde a problemática de la misma área de procesos, entre otros; le permiten al equipo de proyecto especializarse en estas características. Esto beneficia procesos posteriores dentro de un proyecto de software puesto que, por ejemplo, influye positivamente en la calidad de los requisitos obtenidos, la cantidad de no conformidades encontradas durante la validación o la adaptación a los cambios que surjan en el desarrollo. No obstante, los directivos de proyecto deben prestar especial atención a la preparación de personal competente, comunicativo, carismático y con una preparación técnica acorde, puesto que de ello depende la calidad de la elicitación de los requisitos y, como consecuencia, del propio sistema.

En la obtención de los requisitos, los analistas aplican un conjunto de técnicas que son seleccionadas de acuerdo al centro de información: características de las personas, documentos, tipo de negocio, entre otros (...) las técnicas de obtención de requisitos son: cuestionarios, entrevistas, análisis de documentos, métodos basados en objetivos o escenarios, métodos demográficos, análisis de conversaciones, tormenta de ideas, reutilización de requisitos, entre otros. (Pérez Huebe 2005)

La técnica de reutilización de requisitos consiste en el reuso de requisitos partiendo de la idea de que los requisitos que ya han sido capturados para alguna aplicación, pueden ser reusados en la especificación de otra aplicación. (Loucopoulos y Karakostas 1995) En ocasiones hay requisitos que se pueden volver a reutilizar sin ni siquiera alterarlos, pero lo más habitual es encontrar especificaciones que son una buena base para construir los requisitos deseados (Instituto Nacional de Tecnologías de la Comunicación 2008). Esto se debe a que un gran porcentaje de los requisitos que se escriben en un proyecto ya habían sido escritos previamente en otros.

La Universidad de las Ciencias Informáticas (UCI) es una institución de educación superior considerada el mayor centro de desarrollo informático en Cuba. Tiene el propósito de organizar la producción desarrollando proyectos, productos y servicios, garantizando la integración y reutilización de aplicaciones y componentes, de acuerdo con las proyecciones y políticas del país. En su estructura cuenta con 15

centros de desarrollo de software de diferentes dominios de aplicaciones y áreas del conocimiento sobre plataformas libre cumpliendo las políticas de informatización de la sociedad cubana.

Otras características de la universidad y sus centros es la disponibilidad de personal que asuma el rol de analista con una experiencia que permita resultados satisfactorios en la elicitación de requisitos. El personal de los proyectos fundamentalmente se asegura con la incorporación de los estudiantes que se gradúan y que, de no haber realizado su práctica laboral en el Centro, requieren de un período de adaptación en el negocio y la tecnología. Según datos de la Dirección de Recursos Humanos de la UCI del 2014 hasta la actualidad, luego de graduados, los especialistas se mantienen en la institución, como promedio, entre 3 y 4 años. Estos dos hechos repercuten negativamente en la estabilidad de los saberes dentro del proyecto y en el nivel de preparación de los que funjan como analistas (y cualquier otro rol), si los directivos no mantienen una seria estrategia de formación y de transferencia del *know how* del proyecto.

Las técnicas más comunes de la elicitación de requisitos se apoyan en la inteligencia colectiva y en la posibilidad de explotar las potencialidades cognitivas o de habilidades que cada uno posea. Sin embargo, no siempre es posible aplicar esta alternativa en los proyectos de la universidad puesto que los clientes no están disponibles de forma conjunta y/o por parte de los centros no se tiene asignado más que un analista, máximo dos, y con frecuencia, sin la experiencia necesaria en las técnicas, en el negocio o en el propio desempeño del rol. Si a ello se le agrega el hecho de que los requisitos no funcionales tienen más complejidad de captura, conlleva a correr riesgos con grandes probabilidades de ocurrencia que traigan como consecuencia la insatisfacción del cliente, el no cumplimiento de los compromisos establecidos contractualmente, al encarecimiento del proyecto en tiempo y costo y, por ende, al fracaso del proyecto o a la necesidad de hacer más de tres iteraciones en períodos de pruebas.

Teniendo en cuenta la situación anteriormente descrita, se identifica el siguiente problema a resolver: ¿Cómo apoyar la elicitación de requisitos no funcionales en los proyectos que se realizan en la UCI con la utilización de experiencia acumulada en desarrollos previos?

El problema está enmarcado en el objeto de estudio: el proceso de elicitación de requisitos de software.

Para solucionar el problema planteado, se define como objetivo general: Desarrollar una herramienta informática para apoyar la elicitación de requisitos no funcionales, basada en la utilización de experiencia acumulada en desarrollos previos, para los proyectos de la Universidad de las Ciencias Informáticas. Este objetivo limita el área de investigación al campo de acción: El proceso de elicitación de requisitos no funcionales de software basado en la experiencia de proyectos anteriores.

Para dar cumplimiento al objetivo general se plantean las siguientes tareas de investigación:

1. Elaboración de los referentes teóricos de la investigación relacionados con el proceso de elicitación de requisitos no funcionales de software.
2. Fundamentación de las tecnologías que se emplean en el desarrollo de la herramienta.
3. Conceptualización del procedimiento que gestiona la elicitación de los requisitos no funcionales a partir de las características del nuevo proyecto.
4. Diseño de la herramienta siguiendo la metodología de desarrollo fundamentada.
5. Implementación de la herramienta siguiendo las buenas prácticas de desarrollo de software y el diseño obtenido.
6. Validación de la solución empleando técnicas de pruebas que compruebe tanto las funcionalidades como el comportamiento del software.

Los métodos científicos utilizados para desarrollar la investigación fueron, dentro de los teóricos:

- Analítico-sintético: Se emplea para, a partir del análisis de cada una de las partes del objeto de estudio y de las partes de una herramienta informática, conformar cada uno de los resultados de la solución.
- Inductivo-deductivo: Se aplica en la elaboración de conclusiones parciales y finales de la investigación, y para arribar a los resultados derivados de los análisis en las diferentes etapas de la investigación.
- Modelación: Se utiliza para generar los diagramas y tablas que representan el diseño y la arquitectura de la herramienta.

Entre los métodos empíricos:

- Observación: Se utiliza para adquirir conocimiento sobre el campo de acción de la investigación y emplearlo en la conceptualización del procedimiento de elicitación de requisitos no funcionales utilizando la herramienta.
- Entrevista: Aplicada como técnica para capturar información proveniente de analistas de software con experiencia para establecer la teoría sobre cómo se identifican los elementos de un proyecto y cómo se realiza la identificación de requisitos no funcionales en diferentes tipos de proyectos.
- Análisis documental: Se utiliza en la revisión de la literatura especializada para extraer la información necesaria que permita la fundamentación de la investigación y el establecimiento de los requisitos de la herramienta.

Para cumplimentar las tareas de investigación el documento se estructura en tres capítulos, siendo estos:

Capítulo 1. Fundamentación teórica sobre la identificación de requisitos no funcionales y su reutilización: En este capítulo se realiza una revisión bibliográfica del estado actual de la temática en estudio, con el objetivo de caracterizar y profundizar las herramientas, lenguaje y metodología que se requieren para dar cumplimiento al objetivo de la presente investigación y fundamentar el uso de técnicas y métodos propios de la elicitación de requisitos.

Capítulo 2. Descripción de la herramienta para apoyar la reutilización de requisitos no funcionales: Este capítulo describe los pasos del método de reutilización que usa la herramienta y documenta técnicamente su desarrollo estructurando la información de acuerdo a la metodología que se emplee.

Capítulo 3. Implementación y pruebas de la herramienta para la identificación de requisitos no funcionales de software: se valida la herramienta de identificación de requisitos no funcionales, a partir de las pruebas que oriente la metodología seleccionada.



## **Capítulo 1. Fundamentación teórica sobre la identificación de requisitos no funcionales y su reutilización**

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

Por lo antes expuesto, se considera a la IR una de las etapas cruciales en un proyecto de software y comprende la definición de requisitos y la elaboración del modelo conceptual del sistema. (Zapata, Giraldo y Mesa 2010)

El proceso general se estructura en cuatro subprocesos de alto nivel: la evaluación, de si el sistema es útil para el negocio (estudio de viabilidad); el descubrimiento de requisitos (obtención y análisis); la transformación de estos requisitos en formularios de estándar (especificación), y la verificación (validación) de que los requisitos, la cual define el sistema que quiere el cliente. La Figura 1-1 muestra la relación entre estas actividades.

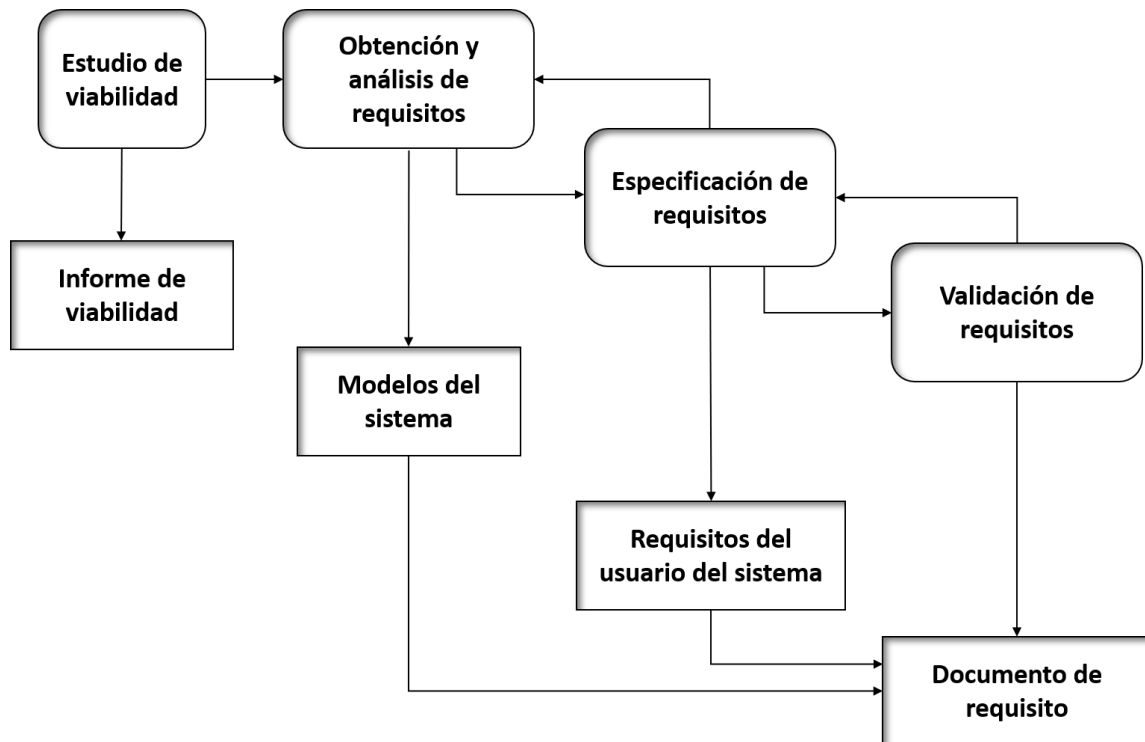


Figura 1-1. Relación de las actividades de la ingeniería de requisitos (Sommerville 2015)

## 1.1 La elicitación de requisitos de software

Esta fase representa el comienzo de cada ciclo de la IR. Involucra a las actividades que sirven para el descubrimiento de los requisitos del sistema. En esta fase los analistas de requisitos deben trabajar junto al cliente para descubrir el problema que el sistema debe resolver, los diferentes servicios que el sistema debe prestar, las restricciones que se pueden presentar, etcétera. Es importante, que la elicitación sea efectiva, ya que la aceptación del sistema dependerá de cuan bien este satisfaga las necesidades del cliente.

La obtención y tratamiento de los requisitos es una tarea a desarrollar minuciosamente por parte de los analistas de software, debido a que los clientes no tienen claras sus necesidades y expectativas sobre el software, incluso la información obtenida por el ingeniero puede variar según la persona a la que consulte o las preguntas que se usen. En el caso de los no funcionales, es importante prestar atención a su identificación y comprensión ya que de esta manera hay una gran posibilidad de alcanzar el éxito en el desarrollo de las aplicaciones de software (Pressman 2010; Gómez 2012).

La elicitación de requisitos es la piedra angular en el desarrollo de proyectos software y tiene un impacto muy alto en el diseño y en las demás fases del ciclo de vida del producto. Si se realiza apropiadamente, puede ayudar a reducir los cambios y las correcciones en los requisitos. Además, la calidad de la

elicitación determina la exactitud de la retroalimentación al cliente acerca de la integridad y validez de los requisitos. Debido a que esta fase es crítica y de alto impacto en el proyecto, es muy importante que la labor de obtención se realice lo más cercano posible a la “perfección”. Teniendo en cuenta las diferentes características de los proyectos de software (Nikual 2011).

Implica un proceso de entendimiento de las necesidades funcionales y las restricciones establecidas por las partes interesadas, las cuales serán entrada para las demás etapas del desarrollo de software. En la elicitación, la inclusión de los requisitos no funcionales depende de las estrategias para gestionar el conocimiento de los interesados (*stakeholders* y elicitadores) con el objetivo de generar productos de software de alta calidad (Buitrón, Flores-Rios y Pino 2018) .

### **1.1.1 Requisitos no funcionales de software**

Muchas son las definiciones de requisitos no funcionales que trata la bibliografía. Los requisitos constituyen descripciones de cómo el sistema debe comportarse, o de una propiedad o atributo del sistema (Sommerville y Sawyer 1997). Uno no funcional puede ser una restricción en el proceso de desarrollo del sistema. Al igual se definen como la 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. (Young 2004, Sawyer y Kotonya 2001)

Los requisitos no funcionales describen las características y aspectos de carácter técnico que el sistema debe poseer. Dichos rasgos distintivos abarcan todo el entorno de funcionamiento del software, desde la infraestructura de procesamiento y conectividad hasta las interfaces visuales, colores y forma en que se debe presentar la información, incluyendo las perspectivas de seguridad informática. (García et al. 2016)

Los requisitos no funcionales se clasifican en (Ver Figura 1-2):

- Requisitos del producto: se encarga de las funciones específicas del sistema de usabilidad, eficiencia, fiabilidad y portabilidad.
- Requisitos organizacionales: se encarga de las funciones específicas del sistema de entrega, estándares e implementación.
- Requisitos externos: se encarga de las funciones específicas del sistema de interoperabilidad, ético y legislativo con privacidad y seguridad.

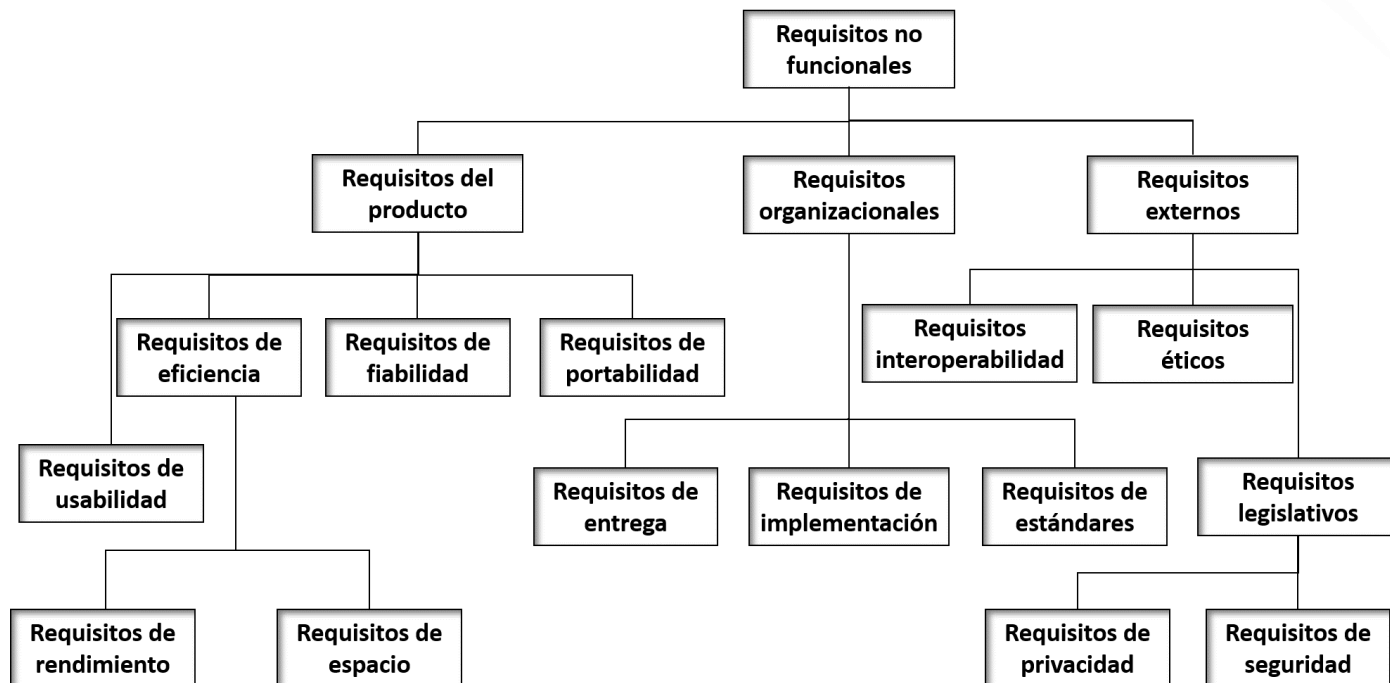


Figura 1-2. Tipos de requisitos no funcionales. (Sommerville 2015)

Para la presente investigación se trabajará con los requisitos no funcionales de producto por la importancia que estos tienen para los diferentes tipos de sistema y porque se concentran en la problemática planteada. Estos requisitos son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este como la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, el rendimiento, la seguridad, entre otros. De forma alternativa, define las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema. (Sommerville 2015) Algunos entendidos denotan su relevancia indicando su criticidad en comparación con los requisitos funcionales particulares. Por ejemplo, si un sistema de vuelo no cumple sus requerimientos de fiabilidad, no se certificará como seguro para su funcionamiento.

La verificación y validación de los requisitos, procesos de la IR, son pobremente tratadas en las bibliografías consultadas. Este proceso en particular tiene una estrecha relación con las características de calidad del software según lo establece el modelo de calidad de producto que abarca cualidad interna y externa del sistema que define la NC ISO/IEC 25010:2016. (Oficina Nacional de Normalización 2016) En este modelo de calidad se establecen ocho (8) características y 31 subcaracterísticas con el objetivo de guiar el desarrollo de los productos de software con la especificación y evaluación de requisitos de calidad, tal como se muestra en la Figura 1-3.



Figura 1-3. Modelo de calidad del producto de la ISO 25010 (Oficina Nacional de Normalización 2016)

Cada característica de calidad debe ser comprobada en el software a través de casos de pruebas que se diseñan en la etapa de análisis del sistema. Estos casos se elaborarán para diferentes tipos de pruebas, técnicas y métodos de forma que se compruebe su validez. Los escenarios de prueba deben ser preparados lo más cercano a la realidad. Es por ello que, para la redacción de los requisitos, muchos analistas de software se mantienen actualizados de los modelos de calidad que certifican sus productos o procesos, si fuera el caso.

### 1.1.2 Técnica de reutilización

En sus inicios, el reuso se limitaba a la reutilización de código fuente, sin tener en cuenta muchos activos que surgen durante todo el proceso del desarrollo software. Sin embargo, gracias al avance en la búsqueda y recuperación de información sobre los repositorios donde se almacenan los activos reutilizables, se produjo una evolución del reuso a niveles mayores de abstracción. Según (Karlsson 1995), un componente reusable es cualquier componente que es desarrollado específicamente para ser usado y, en efecto, es utilizado en más de un contexto. Esto no sólo incluye código sino otros productos del ciclo de vida del sistema, tales como especificaciones, diseños y requisitos. Trae un conjunto de ventajas que se numeran a continuación:

- Reducción de los costos de desarrollo.
- Aumento de la calidad de los productos.
- Patrón de clasificación de tipos de requisitos.
- Aumento de la productividad mediante la mejora de los tiempos en los que se desarrollan los nuevos proyectos.
- Mejoras en las actividades de mantenimiento y soporte de aplicación.

- Mejoras en las actividades de control y planificación por la reducción de desviaciones en los desarrollos.

Los aspectos menos beneficios a tener en cuenta en la reutilización software son:

- Técnicas tradicionales implican una alta inversión inicial, lo que dificulta la amortización de la inversión.
- Nuevas herramientas y cambios en el proceso productivo avivan el “temor al cambio” de los especialistas que, en algunos casos, por falta de confianza en el reuso de software, son detractores estas técnicas.

Como técnica de elicitación de requisitos consiste en seleccionar los productos disponibles en el mercado que más se asemejen a las necesidades de la empresa y aprovecharlos para mejorar la calidad, aumentar la productividad o reducir los costos. De esta manera surge la idea de reuso, utilizar los productos ya existentes para fabricar nuevos productos en vez de desperdiciar esfuerzos creando todo desde cero. (Visure 2018)

La reutilización de los requisitos de software ha recibido una atención importante porque proporciona un soporte sólido para desarrollar software de calidad mediante la obtención y reutilización de los requisitos de software de calidad. (Pacheco et al. 2016)

Este proceso hace que cuantos más objetos sean posibles reutilizar menos recursos se tendrán que gastar. Su sinónimo es reuso y en la presente investigación se denominará de las dos formas reutilización o reuso. A partir de la definición de reuso se puede abordar la técnica de reuso

En las organizaciones, constantemente se dedican esfuerzos a solucionar problemas que, a pesar de ser nuevos para el equipo de trabajo, no son nuevos para el mundo. Reutilizar la experiencia y las lecciones aprendidas por otra persona que alguna vez se encontró en una situación semejante presenta un nuevo reto para el ser humano. Los patrones son una manera de compartir estas buenas ideas que algunos expertos han utilizado con éxito para afrontar los problemas.

La técnica de reuso de requisitos parte de la idea de que los requisitos que ya han sido capturados para alguna aplicación, pueden ser reusados en la especificación de otra aplicación similar (Loucopoulos y Karakostas 1995). En ocasiones hay requisitos que se pueden volver a reutilizar sin ni siquiera alterarlos, pero lo más habitual es encontrar especificaciones que son una buena base para construir los requisitos deseados (INTECO 2008). Entre las razones que consideran su uso se encuentra: el ahorro de tiempo con la consecuente mejora de productividad, el nivel significativo de similitud entre sistemas que pertenecen a una misma área de aplicación y la potencial obtención de mejoras. (Castañeda 2015)

La técnica de reutilización de requisitos emplea métodos que se basan en las clasificaciones de estos de acuerdo a los estándares que apliquen. El método nombrado *Simple REuse of software requiremeNts* (SIREN), por ejemplo, se basa en la utilización de un modelo de proceso en espiral, las plantillas de documentos de requisitos y un repositorio de requisitos reutilizables, que se encuentra organizado por catálogos. Dichos catálogos de requisitos se corresponden con lo que se denominan: (Lasheras et al. 2003)

- ámbito de proyecto o dominios de aplicación “horizontales”. Estos especifican qué sector de la sociedad es objeto de atención en el sistema, dígame: salud, educación, comercio, entre otros. etcétera)
- área de proyecto o dominios de aplicación “verticales”, que representa el negocio que puede ser hospitales, ensayos clínicos, objetos de aprendizaje, etcétera.

Los catálogos se componen de una jerarquía de documentos de especificación, estructurados de acuerdo a estándares de la norma IEEE.

Todo requisito en SIREN tiene un conjunto mínimo, común, de atributos, aunque dependiendo del catálogo puede tener definidos atributos adicionales. A esta forma de requisito se le llama parametrizado y se diferencia en que contiene alguna parte que se tiene que adaptar a cada aplicación o sistema, teniendo que ser instanciados en el momento de la reutilización. (Lasheras et al. 2003) Un ejemplo de requisito es “El tiempo de respuesta es 5 segundos”; al parametrizarlo el requisito se definiría de la siguiente manera: “El tiempo de respuesta es {valor} segundos”, donde la variable valor es igual a 5.

La investigación se guía, para el diseño de la herramienta de identificación de requisitos no funcionales, por lo que plantea el método SIREN utilizando requisitos parametrizados. Los proyectos pueden agruparse por el tipo de proyecto que no es más que la plataforma para la que se desarrolla la aplicación que pueden ser: web, desktop, aplicaciones móviles, entre otras clasificaciones. Esto se debe a que los proyectos del mismo tipo tienden a especificar requisitos no funcionales similares. Además, se agrupan por los ámbitos y las áreas de proyecto.

## **1.2 Herramientas informáticas relacionadas a la gestión e identificación de los requisitos**

Existen herramientas que han sido creados para apoyar a los especialistas en la gestión e identificación de requisitos. En este epígrafe se analizan varios y se indaga fundamentalmente en tres aspectos: la clasificación de los requisitos que identifica y/o gestiona, si utiliza requisitos parametrizados y la técnica de

reutilización de requisitos. Este análisis se realiza con el propósito de determinar sus capacidades de uso en la universidad o de utilización como base de funcionalidades en una nueva solución. Los sistemas son:

CALIBER-Requisite Management (REM): es una herramienta gratuita de Gestión de Requisitos diseñada para soportar la fase de definición e ingeniería de requisitos de un proyecto de desarrollo software. Permite reflejar la correcta captura de requisitos según lo indicado por el cliente obteniendo un documento ordenado y sin perder flexibilidad a la hora de definir el formato estándar de dicho documento. (Durán 2016) En el documento que se genera se puede indicar qué tipo es: documento de requisitos del sistema, documento de análisis del sistema, registro de conflictos y defectos, registro de peticiones de cambio.(Durán 2016) Solo se puede usar en el sistema operativo Windows desde la versión 7 hasta la más actual. El software solo trabaja con al menos unos de estos tipos de navegador: MS Internet Explorer, Firefox, Chrome, Safari. Requiere Microsoft Office 2007, 2010, 2013 (necesario para DocFactory, Export), Office 365 ProPlus, Office 2016. («Caliber | Micro Focus» 2019)

Para su funcionamiento es indispensable tener también: («Caliber | Micro Focus» 2019)

- Explorador de Visual Studio: VS 2010, 2013 (necesario para la trazabilidad de TFS)
- MS Access: 2003, 2007, 2010, 2013 (necesario para Datamart, Exportar)
- Oracle 11g (necesario para Datamart, Exportar)
- MS SQL Server 2008, 2012 (necesario para Datamart, exportar)
- Adobe Flash Player 9.0.47 o posterior (necesario para el tiempo de ejecución de la visualización)

Esta herramienta no es posible emplearla como solución puesto que no permite reutilizar los requisitos y tampoco parametrizarlos. De ella se toman las iniciativas para la creación y guardado de los datos de un proyecto.

DOOR: es una herramienta para la gestión de requisitos donde los considera como objetos y los documentos como módulos. Tiene una orientación basada en objetos, que puede ayudar a reducir costos, aumentar la eficiencia y mejorar la calidad ya que permite optimizar la comunicación, la colaboración y la verificación de requisitos en la organización y en toda la cadena de suministros.

Fue creada por la empresa sueca Telelogic y posteriormente fue adquirida por IBM. DOORS es una base de datos que permite el almacenamiento estructurado y la administración de requisitos, que cuenta con atributos, tanto libres como a elegir en un menú. (IBM-DOORS 2015)

Se puede utilizar en los sistemas operativos GUN/Linux, Solaris y Windows. Desarrollado por Rational Software, cuenta con una serie de herramientas para el análisis de requisitos y permite automatizar



diferentes pasos en el uso de la herramienta con la ayuda de un lenguaje de script llamado DXL, siglas provenientes de DOORS eXtended Language. (IBM-DOORS 2015) Es una aplicación de uso habitual en la industria aeronáutica, de automovilística y ferroviaria. El producto *Simulink Requirements* de *MathWorks* le permite vincular las especificaciones de diseño y los requisitos de las PUERTAS directamente a sus implementaciones en MATLAB, Simulink y Stateflow. (IBM-Rational 2015) Presenta como limitación para la investigación que no permite el reúso de requisitos no funcionales y ni la parametrización. Necesita la instalación de MATLAB, Simulink y Stateflow para algunas funcionalidades. De esta herramienta se analizan para la investigación el mecanismo de gestión de cambios de la selección de requisitos no funcionales.

Rational Requirements Composer: esta herramienta es creada por IBM, permite a los equipos definir, gestionar y presentar requisitos en un proyecto de desarrollo del ciclo de vida. Es una herramienta basada en la web que da soporte a metodologías de desarrollo iterativas, en cascada y fácilmente escalables mediante procesos de requisitos ligeros. *Rational Requirements Composer* ayuda a gestionar el ciclo de vida de la información de los requisitos, mediante la implementación de un enfoque amplio y flexible que permita a los equipos colaborar, clarificar y llegar a un acuerdo sobre los requisitos. (IBM-RRC 2016)

La herramienta para usarla requiere del sistema operativo Windows donde se instala un *plug-in* en el navegador que solo puede ser:

- Mozilla FireFox (FF) (recomendado)
- Microsoft Internet Explorer

No es factible su uso debido a que, como las anteriores, no realiza el reúso de requisitos ni la parametrización. De ella se asume la forma de guardar directamente los requisitos en la base de datos.

### **1.3 Metodologías de desarrollo**

Una metodología de desarrollo de software, consiste principalmente en hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo. Regularmente este tipo de metodología, tienen la necesidad de venir documentadas, para que los programadores que están dentro de la planeación del proyecto, comprendan perfectamente la metodología y en algunos casos el ciclo de vida del software que se pretende seguir.(Canós y Letelier 2012)

La UCI emplea su propia metodología de desarrollo para la actividad productiva. Dicha definición se basa en una variación de la metodología de Proceso Unificado Ágil (AUP por sus siglas en inglés), en unión con el modelo CMMI-DEV v 1.3. Se decide emplearla como guía para la construcción de la herramienta puesto que formará parte de los sistemas creados por y para la institución.

El Proceso Unificado Ágil (*Agile Unified Process*, AUP) es una versión simplificada del Proceso Unificado Racional (RUP). Este describe de una manera simple y fácil de entender, la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. La UCI le ha realizado modificaciones con el fin de adaptarlo al ciclo de vida definido para la actividad productiva de dicha institución; de las cuatro fases que encierra la metodología AUP se simplificaron a: (Sánchez 2015)

- Inicio: 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, costo y decidir si se ejecuta o no el proyecto.
- Ejecución: en esta fase se recogen las actividades que desarrolla AUP de elaboración, construcción y transición. Se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura.
- Cierre: En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Al ser esta investigación un trabajo de diploma, se trabajará basado en las actividades de la fase de ejecución y que documentan el proceso de desarrollo de la herramienta.

Para el ciclo de vida de los proyectos de la UCI la metodología AUP-UCI consta de ocho (8) disciplinas, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos, Análisis y Diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se considera cada uno de estos flujos como disciplinas. Se mantiene la disciplina Implementación y, en el caso de Prueba, se desagrega en tres (3) disciplinas: Pruebas internas, de liberación y aceptación y la disciplina Despliegue se considera opcional.

El escenario de la disciplina Requisitos que se aplica es el Escenario 2 para proyectos que modelen el negocio con modelo conceptual. Este plantea:

Escenario No2: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda este escenario para proyectos donde el objetivo primario es la gestión y presentación de información. (Sánchez 2015)

## **1.4 Tecnologías y herramientas**

A continuación, se describen las principales herramientas y tecnologías seleccionadas para ser utilizadas en el proceso de desarrollo del software, las cuales son de software libre.

### **1.4.1 Lenguaje de programación**

El lenguaje de programación es aquella estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora. (Horstmann y Budd 2009) En la implementación de la herramienta se decide emplear:

Java 8.4: por ser robusto y multiplataforma. 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 es interpretado por una máquina virtual. (Horstmann 2007)

El lenguaje Java está diseñado para permitir el desarrollo de aplicaciones portátiles, de elevado rendimiento, para el más amplio rango de plataformas informáticas posible. Al poner a disposición de todo el mundo, aplicaciones en entornos heterogéneos, las empresas pueden proporcionar más servicios y mejorar la productividad, las comunicaciones y colaboración del usuario final y reducir drásticamente el costo de propiedad, tanto para aplicaciones de usuario, como de empresa. Java tiene un alto potencial puesto que permite:

- Escribir software en una plataforma y ejecutarla virtualmente en otra.
- Crear programas que se puedan ejecutar en un explorador y acceder a servicios Web disponibles.
- Desarrollar aplicaciones de servidor para foros en línea, almacenes, encuestas, procesamiento de formularios HTML y mucho más.
- Combinar aplicaciones o servicios que utilizan el lenguaje Java para crear aplicaciones o servicios con un gran nivel de personalización.

### **1.4.2 Tecnologías y herramientas**

Las tecnologías y herramientas empleadas completan el trabajo del lenguaje de programación empleado. De esta manera se trabaja con varios marcos de trabajos, entornos de desarrollo entre otros que se integran con Java. Las herramientas y tecnologías son:

JavaFX: este es un conjunto de paquetes de gráficos y multimedia que permite a los desarrolladores diseñar, crear, probar, depurar y desplegar aplicaciones de cliente, que operan constantemente a través de diversas plataformas.” (Oracle 2014) Escrito como una API Java, el código de aplicación JavaFX puede hacer referencia a APIs de cualquier biblioteca Java. Por ejemplo, las aplicaciones JavaFX pueden utilizar

bibliotecas de API de Java para acceder a las capacidades del sistema nativo y conectarse a aplicaciones de middleware basadas en servidor.

El aspecto de las aplicaciones JavaFX se puede personalizar. Las hojas de estilo en cascada (CSS) separan la apariencia y el estilo de la implementación para que los desarrolladores puedan concentrarse en la codificación. Los diseñadores gráficos pueden personalizar fácilmente la apariencia y el estilo de la aplicación a través del CSS. Si tiene experiencia en diseño web, o si desea separar la interfaz de usuario (UI) y la lógica de *back-end*, puede desarrollar los aspectos de presentación de la UI en el lenguaje de scripting FXML y utilizar código Java para la lógica de la aplicación. A medida que diseña la interfaz de usuario, Scene Builder crea marcas FXML que se pueden portar a un entorno de desarrollo integrado (IDE) para que los desarrolladores puedan añadir la lógica empresarial. (Oracle 2016) Facilita notablemente la construcción de las interfaces de aplicaciones que serán construidas con la herramienta Scene Builder.

JUnit: es un framework Java que permite la realización de la ejecución de clases de manera controlada, para poder comprobar que los métodos realizan su cometido de forma correcta. (JUnit 2018) Contiene un conjunto de librerías que facilitan la realización de dichas pruebas, necesarias para verificar que los métodos implementados se desempeñen de forma correcta.

Scene Builder: es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de las aplicaciones JavaFX, sin necesidad de programación. Los usuarios pueden arrastrar y soltar los componentes de interfaz de usuario a un área de trabajo y modificar sus propiedades. Se aplican las hojas de estilo (css), y el código FXML se genera automáticamente en segundo plano según los elementos que se vayan creando. El resultado es un archivo FXML que a continuación se puede combinar con un proyecto de Java mediante la unión de la interfaz de usuario a la lógica de la aplicación.” (Oracle 2018)

IDE: NetBeans 8.0.2: se utiliza como entorno de desarrollo integrado (IDE): teniendo en cuenta que es una solución muy completa para programar en Java, aunque soporta otros lenguajes como C/C++, JavaScript, Ruby, Groovy, Python y PHP. El proyecto de NetBeans está apoyado por una comunidad de desarrolladores y ofrece una amplia documentación y recursos de capacitación y consta de una gran cantidad de módulos. Además, puede ser usado tanto por programadores con poca experiencia como por expertos y permite trabajar sobre el mismo código a más de un programador. Puede ejecutarse en Windows, OS X, Linux, Solaris y otras plataformas que soportan una Java Virtual Machine compatible. Existe un número importante de módulos para extender el NetBeans. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

Algunas de las características son:

- Integración MySQL más ajustada y una mejor manera de compartir librerías entre proyectos dependientes
- El aclamado soporte para Ruby/JRuby ha sido mejorado con un nuevo editor de soluciones rápidas (Quick Fix)
- Un administrador para la plataforma Ruby
- Soporte para depuración rápida en JRuby
- Registro de servidores MySQL (NetBeans 2018)

JFoenix: es una librería java de código abierto, que implementa Google Material Design utilizando componentes de java, los cuales tienen un diseño más amigable para el usuario final. (JetBrains 2017) Se prevé emplear para el diseño de las interfaces visuales de la herramienta.

#### **1.4.3 Herramienta CASE y lenguaje de modelado**

Visual Paradigm 8.0: es una herramienta CASE que propicia la modelación de los artefactos de la herramienta. Tiene una amplia gama de funcionalidades para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Es una herramienta con disponibilidad en múltiples plataformas y soporta varios Sistemas Gestores de Bases de Datos como Oracle, Microsoft SQL Server, MySQL PostgreSQL y SQLite. También permite el uso de las distintas metodologías propias de la Ingeniería de Software. (Visual Paradigm 2018)

UML 2.0: versión del Lenguaje Unificado de Modelado que estandariza la creación de esquemas, diagramas y documentaciones relativas al desarrollo de software. Es ampliamente empleado a nivel internacional.

#### **1.4.4 Sistema de gestión de base de datos**

Un sistema de gestión de base de datos (SGBD) es un conjunto de programas que permiten la creación de base de datos y proporciona herramientas para añadir, borrar, modificar, y eliminar datos, además de mantener la integridad de estos. Para la base de datos de la herramienta se prevé emplear a:

PostgreSQL 9.3: por ser un potente SGBD objeto relacional de código abierto, tiene soporte completo para claves foráneas, vistas y disparadores. Incluye la mayoría de los tipos de datos de SQL 2008 incluyendo integer, boolean y varchar entre otros. También soporta el almacenamiento de objetos binarios grandes como imágenes, sonido o video. Algunas de las características de PostgreSQL son: (PostgreSQL 2018)

- Arquitectura cliente servidor con un amplio rango de drivers
- Diseño de alta concurrencia donde lectores y escritores no se bloquean
- Altamente confiable y extensible para muchos tipos de aplicación

- Optimizador de consultas sofisticado
- Incluye herencia entre tablas

PgAdmin III: es una herramienta de código abierto para la administración de bases de datos PostgreSQL y derivados (EnterpriseDB, PostgresPlus, Advanced Server y GreenplumDatabase. Responde a las necesidades de la mayoría de los usuarios, desde escribir simples consultas SQL hasta desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL y hace simple la administración. Está disponible en más de una docena de lenguajes y para varios sistemas operativos, incluyendo Microsoft Windows, Linux, FreeBSD, Mac OSX y Solaris. Soporta versiones de servidores 7.3 y superiores. Versiones anteriores a 7.3 deben usar PgAdmin II. (pgadmin 2018)

## **1.5 Conclusiones del capítulo**

En este capítulo se fundamentan los métodos, conceptos y teorías propias del objeto de estudio en el que se enmarca la investigación que apoyarán la concepción y diseño de la herramienta de elicitación de requisitos no funcionales. Se obtienen las siguientes conclusiones:

- Se determina que las herramientas analizadas no cumplen con los elementos requeridos y se analiza el empleo de iniciativas usadas en ellas, para la construcción de la solución objeto de la investigación.
- Como herramientas y tecnologías para desarrollar la solución se seleccionó como entorno de desarrollo integrado a utilizar NetBeans 8.0.2, así como el lenguaje de programación oficial Java y JUnit para la validación; así como gestor de base de datos PgAdmin III y PostgreSQL 9.3 y la herramienta de modelado seleccionada fue Visual Paradigm 8.0 con el lenguaje UML 2.0 capaz de representar el ciclo de vida completo de software.
- El desarrollo de la herramienta está guiado por la metodología AUP en su variante UCI, escenario 2, respetando las políticas del proceso de desarrollo de software empleadas en la universidad.

## Capítulo 2. Descripción de la herramienta para apoyar la elicitación de requisitos no funcionales

La solución que se propone se basa en el método SIREN. En este método, de acuerdo a (Cáceres Saldaña 2014), se utilizan dos enfoques: el desarrollo para reutilización y el desarrollo con reutilización. Cada enfoque se corresponde con los siguientes procesos respectivamente:

- Indexación: procedimiento que debe seguir el analista para transformar un conjunto de requisitos para que puedan ser almacenados en un repositorio con el objetivo final de registrar ordenadamente toda la información de los requisitos para incluirlos en el repositorio.
- Recuperación: proceso conformado por una serie de etapas que debe seguir el analista para reutilizar los requisitos del catálogo y utilizarlos en el proyecto.

En el proceso de indexación se realizan cuatro (4) actividades que son:

1. Identificación de los posibles requisitos para almacenarlo en el repositorio de requisitos.
2. Adaptación del requisito para lograr la parametrización y clasificación de los requisitos identificados.
3. Verificación de la calidad de la parametrización del requisito. Si no es suficiente se vuelve a aplicar la actividad de adaptación (paso 3), sino se aplica la actividad de almacenamiento (paso 4).
4. Almacenamiento del requisito en el repositorio de requisitos.

En el proceso de recuperación se contienen tres (3) actividades que son:

1. Consulta para identificar los requisitos no funcionales a partir de las características del proyecto.
2. Selección de los requisitos que más se adecuan en el proyecto que se esté analizando.
3. Adecuación de los requisitos al proyecto en el cual se va emplear.

Estas actividades se representan gráficamente en la Figura 2-1.

Se consideran características de un proyecto al tipo de proyecto en el que se clasifica, el ámbito y el área en el que se enmarca.

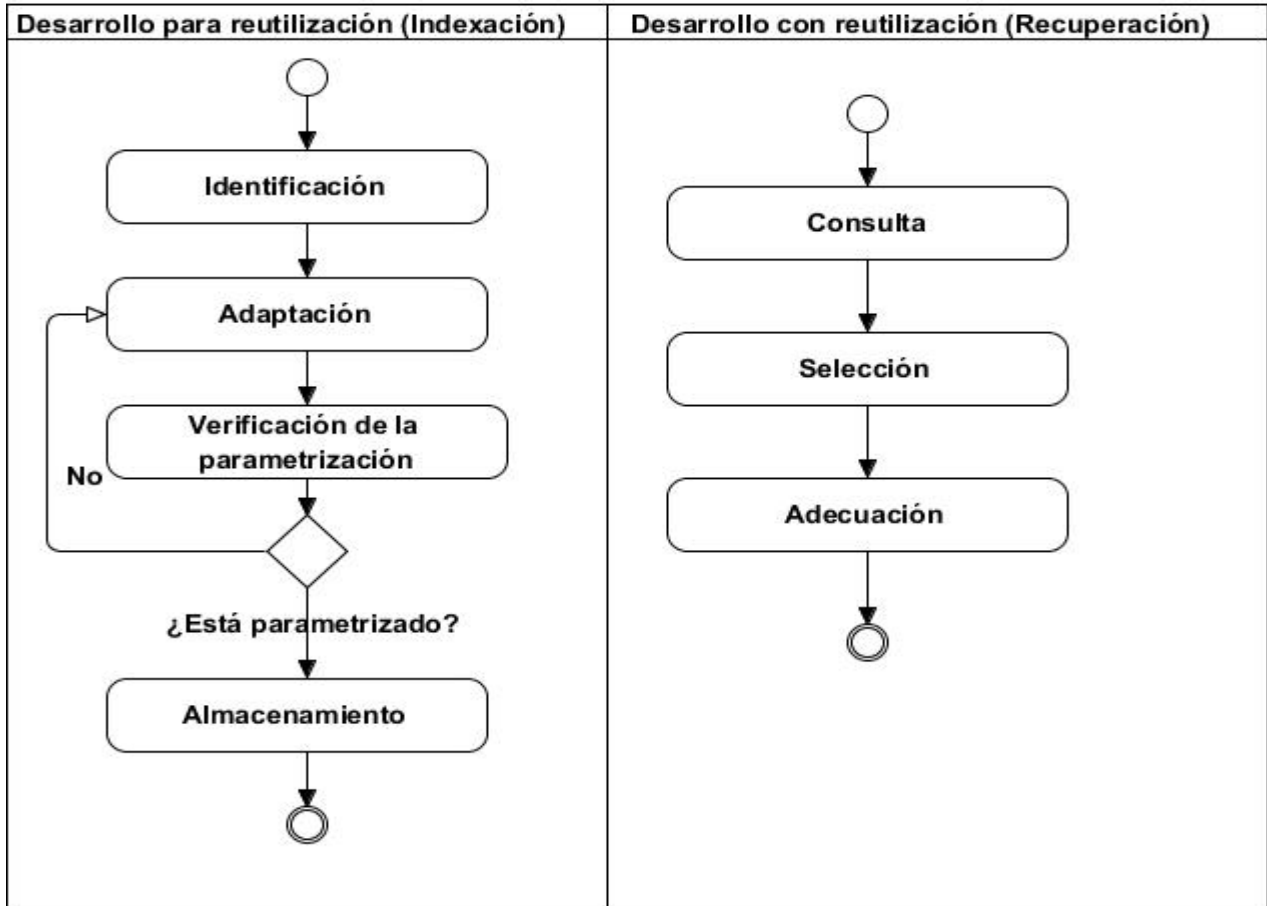


Figura 2-1. Representación gráfica del método de reutilización de requisitos SIREN. Fuente: Elaboración propia

## 2.1 Modelado del negocio

El modelado de negocio, primera disciplina de la metodología AUP-UCI, se realiza en este trabajo utilizando un modelo conceptual, propio del escenario escogido y porque se considera importante representar los conceptos que intervienen en el contexto del sistema, no así la interacción de los usuarios con este. En la Figura 2-2 se muestra el diagrama de este modelo y a continuación la descripción de los objetos.



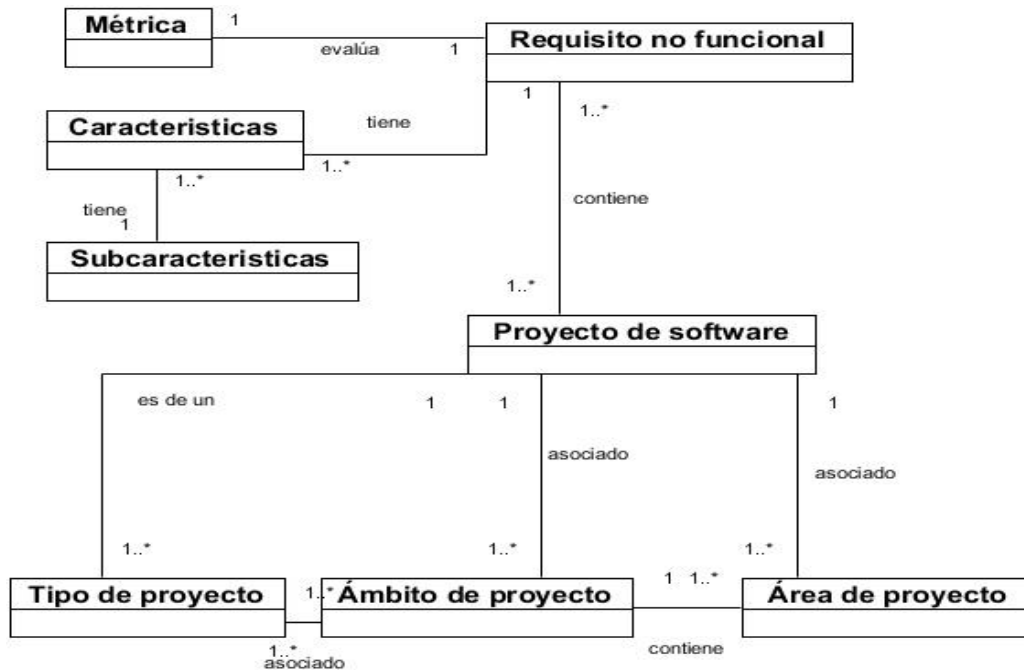


Figura 2-2. Modelo conceptual del negocio. Fuente: Elaboración propia

Descripción de los objetos del modelo conceptual

**Requisito no funcional:** son los requisitos que rigen el comportamiento del sistema el cual contiene características y subcaracterísticas.

**Métrica:** Criterio de calidad por el que se evalúa un requisito no funcional.

**Característica:** Clasificación de los requisitos de acuerdo al estándar de calidad ISO 25010. Ver Figura 1-3.

**Subcaracterística:** Clasificación de la característica de acuerdo al estándar de calidad ISO 25010. Ver Figura 1-3.

**Tipo de proyecto:** Tipo de plataforma para la cual se desarrolla el proyecto: escritorio, web, móvil, sistema operativo, entre otros que se definan.

**Ámbito de proyecto:** Clasifica al proyecto de acuerdo al sector de la sociedad para el que se realiza.

**Área de proyecto:** Identifica el negocio, dentro del ámbito de proyecto definido, al que se vincula el proyecto.

**Proyecto de software:** Conjunto de actividades a realizar en un tiempo limitado, con un objetivo y costo determinado.

## 2.2 Identificación de requisitos

Los requisitos funcionales que se definen en la investigación para el desarrollo de la herramienta son los que se muestran a continuación:

- RF 1 Insertar requisito no funcional.
- RF 2 Editar requisito no funcional.
- RF 3 Eliminar requisito no funcional.
- RF 4 Insertar proyecto.
- RF 5 Editar proyecto.
- RF 6 Eliminar proyecto.
- RF 7 Insertar métrica.
- RF 9 Editar métrica.
- RF 10 Eliminar métrica
- RF 11 Insertar características y subcaracterística.
- RF 12 Editar característica y subcaracterística.
- RF 13 Eliminar característica y subcaracterística.
- RF 14 Insertar usuario.
- RF 15 Editar usuario.
- RF 16 Eliminar usuario.
- RF 17 Insertar centro de producción.
- RF 18 Editar centro de producción.
- RF 19 Eliminar centro de producción.
- RF 20 Insertar tipo de proyecto.
- RF 21 Editar tipo de proyecto.
- RF 22 Eliminar tipo de proyecto.
- RF 23 Insertar ámbito de proyecto.

- RF 24 Editar ámbito de proyecto.
- RF 25 Eliminar ámbito de proyecto.
- RF 26 Insertar área de proyecto.
- RF 27 Editar área de proyecto.
- RF 28 Eliminar área de proyecto.
- RF 29 Identificar requisitos no funcionales
- RF 30 Reutilizar requisitos no funcionales
- RF 31 Exportar documento de requisitos no funcionales a PDF

Los requisitos no funcionales que se tienen en cuenta para el desarrollo de la herramienta son:

- **Usabilidad**

RNF 1: Los grupos de botones y vínculos deben organizarse por funcionalidad, con el objetivo de facilitar al usuario la interacción con el software.

RNF 2: Los mensajes para interactuar con los usuarios y los de error deben ser lo suficientemente informativos, en idioma español y no deben revelar información interna.

- **Software**

RNF 3: La computadora donde se ejecute la herramienta debe tener instalada una versión de la máquina virtual de Java 8.0.91 o superior.

- **Hardware**

RNF 4: Características de hardware de los ordenadores donde se desee instalar la herramienta:

- ✓ RAM: 2 GB
- ✓ Procesador: Intel(R) Dual Core CPU @ 2.50 GHz
- ✓ Espacio en disco disponible: 500 Mb

RNF 5: Para la visualización de la aplicación se deben emplear monitores con resolución mínima de 1024x768.

### **2.2.1 Casos de uso del sistema**

Para encapsular los requisitos se emplean casos de uso del sistema. En estos intervienen los actores que no son más que sujetos, objetos o un mismo sistema que interactúa con este de forma externa, que se

relaciona con éste ya que solicita sus funcionalidades. Los actores de la solución son los descritos en la tabla.

Tabla 2-1 Actores del sistema. Fuente: Elaboración propia

<b>Actor</b>	<b>Descripción</b>
<b>Analista</b>	Es la persona autenticada en el sistema y que accede con el fin de insertar nuevos proyectos, realizar la identificación y reutilización de los requisitos no funcionales o insertar nuevos requisitos no funcionales.
<b>Administrador</b>	Es la persona autenticada en el sistema y que accede con el fin de insertar, modificar, eliminar, habilitar o deshabilitar los analistas del sistema.

El diagrama de casos de uso documenta el comportamiento de un sistema desde el punto de vista del usuario. Se utiliza para describir las funcionalidades de un software y documentar su comportamiento. Los casos de uso del sistema definidos para la solución son:

- CU 1 Gestionar el repositorio de requisitos
- CU 2 Gestionar proyectos
- CU 3 Gestionar métricas.
- CU 4 Gestionar características y subcaracterísticas
- CU 5 Gestionar usuarios
- CU 6 Gestionar centros de producción
- CU 7 Gestionar tipos de proyectos.
- CU 8 Gestionar ámbito de proyectos
- CU 9 Gestionar áreas de proyecto
- CU 10 Identificar requisitos no funcionales
- CU 11 Reutilizar requisitos no funcionales
- CU 12 Exportar documento de requisitos no funcionales del proyecto a PDF

El diagrama de casos de uso del sistema para esta solución se representa en la Figura 2-3.

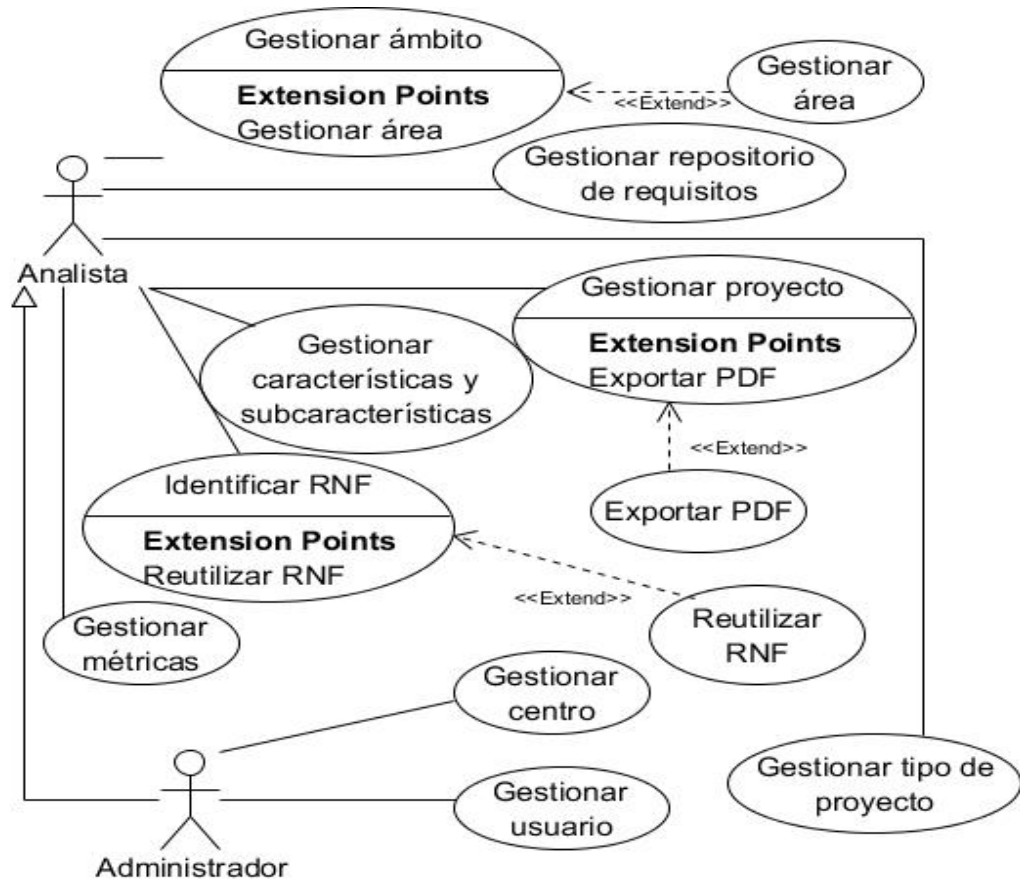


Figura 2-3 Casos de uso del sistema. Fuente: Elaboración propia

### Descripción de casos de usos del sistema

Tabla 2-2 Descripción del CU 1. Gestionar el repositorio de requisitos. Fuente: Elaboración propia

<b>Objetivo</b>	Gestionar los requisitos, teniendo la opción de crearlos, modificarlos y eliminarlos en cualquier momento.
<b>Actores</b>	Analista(Inicia): crea, modifica y elimina los requisitos en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción Repositorio de requisitos, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar un requisito.
<b>Complejidad</b>	Media.
<b>Prioridad</b>	Alta.
<b>Precondiciones</b>	El analista debe estar autenticado en el sistema.
<b>Postcondiciones</b>	Se creó, se editó, se eliminó correctamente y se visualizó el requisito.
<b>Flujo de eventos</b>	

<b>Flujo básico Gestionar requisitos</b>		
	<b>Actor</b>	<b>Sistema</b>
1.	Selecciona la opción Gestionar repositorio.	
2.		a) Si el usuario decide crear un requisito. Ver Sección 1: Registrar requisito.
		b) Si el usuario decide modificar un requisito. Ver Sección 2: Modificar requisito.
		c) Si el usuario decide eliminar un requisito. Ver Sección 3: Eliminar requisito.
3.		Finaliza el caso de uso.
<b>Sección 1: “Registrar requisito”</b>		
<b>Flujo básico: Registrar requisito</b>		
	<b>Actor</b>	<b>Sistema</b>
1	Selecciona la opción que le permite “Registrar” un nuevo requisito.	
2		<p>En la interfaz limpia los campos para permitir introducir los nuevos datos del requisito:</p> <ul style="list-style-type: none"> <li>• Requisito</li> <li>• Característica</li> <li>• Subcaracterística</li> <li>• Métrica</li> </ul> <p>Activa las opciones:</p> <ul style="list-style-type: none"> <li>• “Aceptar”</li> <li>• “Cancelar”</li> </ul>
3	Selecciona e introduce los datos deseados.	
4	Selecciona el botón “Aceptar”.	
5		Verifica que se hayan introducido los campos obligatorios.
6		Verifica que los datos introducidos sean correctos.
7		Se muestra un mensaje de información “El requisito ha sido creado correctamente”.
8		Actualiza el listado de requisitos con el nuevo elemento.

9		Finaliza el caso de uso.
<b>Flujos alternos Registrar requisito</b>		
<b>Nº Evento 2: Cancelar</b>		
	<b>Actor</b>	<b>Sistema</b>
1	Presiona el botón "Cancelar".	
2		Cancela la operación de Registrar requisito.
3		Desbloquea la tabla para mostrar los requisitos almacenados.
<b>Nº Evento 5: Campos obligatorios vacíos</b>		
1		Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".
<b>Nº Evento 6: Datos incorrectos</b>		
1		Muestra un mensaje de error: "Los datos introducidos son incorrectos".
<b>Sección 2: "Modificar requisito"</b>		
<b>Flujo básico: Modificar requisito</b>		
<b>No.</b>	<b>Actor</b>	<b>Sistema</b>
1.	Selecciona la opción que le permite "Modificar" los datos de un requisito.	
2.		Permite modificar los datos del reporte seleccionado: <ul style="list-style-type: none"> <li>• Requisito</li> <li>• Característica</li> <li>• Subcaracterística</li> <li>• Métrica</li> </ul> Activa las opciones: <ul style="list-style-type: none"> <li>• "Aceptar"</li> <li>• "Cancelar"</li> </ul>
3.	Modifica los datos deseados.	
4.	Selecciona el botón "Aceptar".	
5.		Verifica que se hayan introducido los campos obligatorios.
6.		Verifica que los datos introducidos sean correctos.
7.		Se muestra un mensaje de información "El requisito se ha actualizado correctamente".
8.		Actualiza el listado de requisitos con el nuevo elemento.

9.		Finaliza el caso de uso.
<b>Flujos alternos Modificar requisito</b>		
<b>Nº Evento 2: Cancelar</b>		
	<b>Actor</b>	<b>Sistema</b>
1.	Presiona el botón "Cancelar".	
2.		Cancela la operación de Registrar requisito.
3.		Desbloquea la tabla para mostrar los requisitos almacenados.
<b>Nº Evento 5: Campos obligatorios vacíos</b>		
1.		Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".
<b>Nº Evento 6: Datos incorrectos</b>		
1.		Muestra un mensaje de error: "Los datos introducidos son incorrectos".
<b>Sección 3: "Eliminar requisito"</b>		
<b>Flujo básico: Eliminar requisito</b>		
	<b>Actor</b>	<b>Sistema</b>
1.	Selecciona la opción que le permite "Eliminar" el requisito seleccionado.	
2.		Muestra un mensaje de confirmación "¿Seguro que desea eliminar el requisito seleccionado?" Y permite: <ul style="list-style-type: none"> <li>• "Aceptar"</li> <li>• "Cancelar"</li> </ul>
3.	Selecciona el botón "Aceptar".	
4.		Elimina el requisito seleccionado.
5.		Se muestra un mensaje de información "El requisito ha sido eliminado correctamente".
6.		Actualiza el listado de requisitos.
<b>Flujos alternos Modificar requisito</b>		
<b>Nº Evento 3: Cancelar</b>		
	<b>Actor</b>	<b>Sistema</b>
1.	Presiona el botón "Cancelar".	
2.		Cancela las operaciones realizadas sobre el requisito.
<b>Relaciones</b>	<b>CU incluidos</b>	No aplica



	<b>CU extendidos</b>	No aplica
<b>Requisitos no funcionales</b>		
<b>Asuntos pendientes</b>		

Tabla 2-3 Descripción del CU 10 Identificar requisitos no funcionales. Fuente: Elaboración propia

<b>Objetivo</b>	Identificar los requisitos no funcionales para un proyecto según sus características.	
<b>Actores</b>	Analista (Inicia).	
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción Identificar RNF, la herramienta muestra una interfaz, donde permite seleccionar las características del proyecto.	
<b>Complejidad</b>	Alta	
<b>Prioridad</b>	Alta	
<b>Precondiciones</b>	El analista debe estar autenticado en el sistema.	
<b>Postcondiciones</b>	Se identificaron todos los requisitos dadas las características introducidas.	
<b>Flujo de eventos</b>		
<b>Flujo básico Identificar RNF</b>		
	<b>Actor</b>	<b>Sistema</b>
1	Selecciona la opción "Identificar RNF".	
2		Muestra una interfaz para seleccionar las características del proyecto: <ul style="list-style-type: none"> <li>• Tipo de proyecto</li> <li>• Ámbito de proyecto</li> <li>• Área de proyecto</li> <li>• Subcaracterística</li> <li>• Métrica</li> </ul> Permite la opción: "Aceptar"
3	Selecciona los datos asociados a cada una de las características del proyecto para la identificación de los requisitos no funcionales.	
4	Selecciona el botón "Aceptar".	
5		Verifica que se hayan seleccionado al menos un dato de

		cada característica.
6		Reutilizar RNF. Ver CU Reutilizar requisito no funcional..
7		Finaliza el caso de uso.
<b>Flujos alternos Identificar RNF</b>		
<b>Nº Evento 5: Seleccionar la menos un dato de cada característica</b>		
	<b>Actor</b>	<b>Sistema</b>
1		Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".
<b>Relaciones</b>	<b>CU incluidos</b>	No aplica
	<b>CU extendidos</b>	Reutilizar RNF. Ver CU Reutilizar requisito no funcional
<b>Requisitos no funcionales</b>		
<b>Asuntos pendientes</b>		

Tabla 2-4 Descripción del CU11 Reutilizar requisitos no funcionales. Fuente: Elaboración propia

<b>Objetivo</b>	Reutilizar los requisitos no funcionales para un proyecto según los requisitos identificados.	
<b>Actores</b>	Analista(Inicia)	
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en la interfaz "Identificar RNF" la opción "Aceptar", la herramienta muestra una interfaz, donde permite seleccionar las el proyecto donde se reutilizarán los requisitos y permite seleccionar cuáles serán los requisitos a utilizar.	
<b>Complejidad</b>	Alta	
<b>Prioridad</b>	Alta	
<b>Precondiciones</b>	El analista debe estar autenticado en el sistema.	
<b>Postcondiciones</b>	Se reutilizaron todos los requisitos necesarios para el proyecto en cuestión.	
<b>Flujo de eventos</b>		
<b>Flujo básico Identificar RNF</b>		
	<b>Actor</b>	<b>Sistema</b>

1.	Selecciona la opción "Aceptar" del CU Identificar RNF.	
2.		<p>Muestra una interfaz para seleccionar el proyecto donde se reutilizarán los requisitos identificados y un listado con los requisitos identificados mostrando los datos de:</p> <ul style="list-style-type: none"> <li>• Requisito</li> <li>• Valor mínimo de la métrica</li> <li>• Valor máximo de la métrica</li> <li>• Valor promedio de la métrica</li> <li>• Métrica</li> </ul> <p>Permite la opción:</p> <ul style="list-style-type: none"> <li>• "Agregar"</li> </ul>
3.	Selecciona el proyecto donde se reutilizarán los requisitos no funcionales.	
4.		<p>Se activan las opciones:</p> <ul style="list-style-type: none"> <li>• "Agregar"</li> </ul>
5.	Selecciona el requisito que desea agregar al proyecto.	
6.	Se selecciona el botón "Agregar".	
7.		<p>Se activan los campos para introducir los valores particulares del requisito para el proyecto:</p> <ul style="list-style-type: none"> <li>• Valor de la métrica</li> <li>• Prioridad</li> <li>• No. Requisito</li> </ul>
8.	Introduce los datos deseados para insertar el requisito al proyecto.	
9.	Selecciona el botón "Aceptar".	
10.		Verifica que se hayan introducido los campos obligatorios.
11.		Verifica que los datos introducidos sean correctos.
12.		Actualiza el listado de requisitos del proyecto.
13.		Finaliza el caso de uso.
<b>Flujos alternos Reutilizar RNF</b>		
<b>Nº Evento 10: Campos obligatorios vacíos</b>		

	Actor	Sistema
1.		Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".
<b>Nº Evento 10: Datos incorrectos</b>		
1.		Muestra un mensaje de error: "Los datos introducidos son incorrectos".
Relaciones	CU incluidos	No aplica
	CU extendidos	No aplica
Requisitos no funcionales		
Asuntos pendientes		

### 2.3 Análisis y diseño

En la disciplina Análisis y diseño es imprescindible definir la arquitectura sobre la que se sustentan los requisitos identificados. En este caso se utiliza una arquitectura tres capas, con el patrón modelo-vista-controlador (MVC) que se representa en la Figura 2.4. Se selecciona este patrón con el objetivo de reducir el esfuerzo de programación en la implementación del sistema e independizar cada parte del sistema de forma que la modificación de una capa sea irrelevante para las otras. A partir del uso de marcos de trabajo basados en el patrón MVC se puede lograr una mejor organización del trabajo y mayor especialización de los desarrolladores y diseñadores (Sellarès 2011).

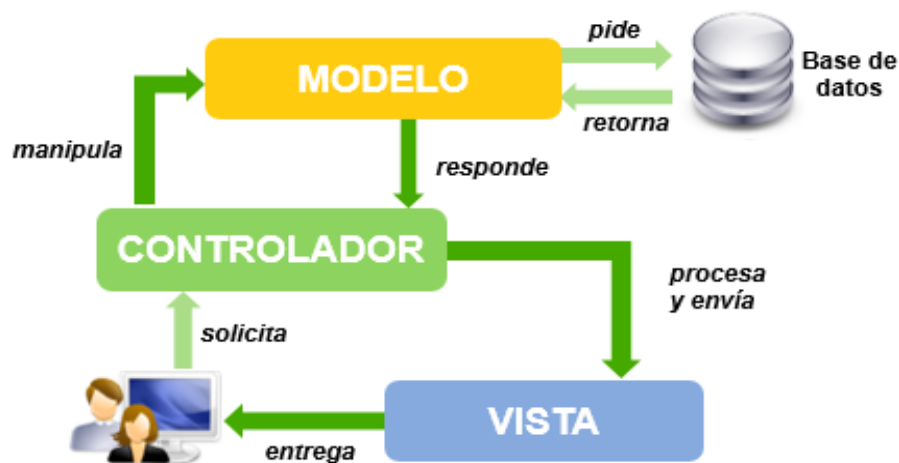


Figura 2-4: Arquitectura MVC. Fuente: Elaboración propia

De acuerdo a las partes del MVC que propone (Sellarès 2011) se establece que:

- El modelo: contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La vista: o interfaz de usuario, que compone la información que se envía al usuario o al administrador, y los mecanismos de interacción con ellos.
- El controlador: actúa como intermediario entre el modelo y la vista gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

Para el desarrollo de la propuesta de solución se establecen las clases controladoras FrmGestionarTipoProyectoController, FrmDetalleRasgoController entre otras. Estas clases se comunican con las interfaces correspondiente de forma que obtienen o envían información que pueden almacenar o consultar de la capa de acceso a datos. Un ejemplo de esta comunicación en la solución que se describe se representa en la Figura 2-5. El resto de las clases que intervienen en el modelo se representan en el modelo de datos.

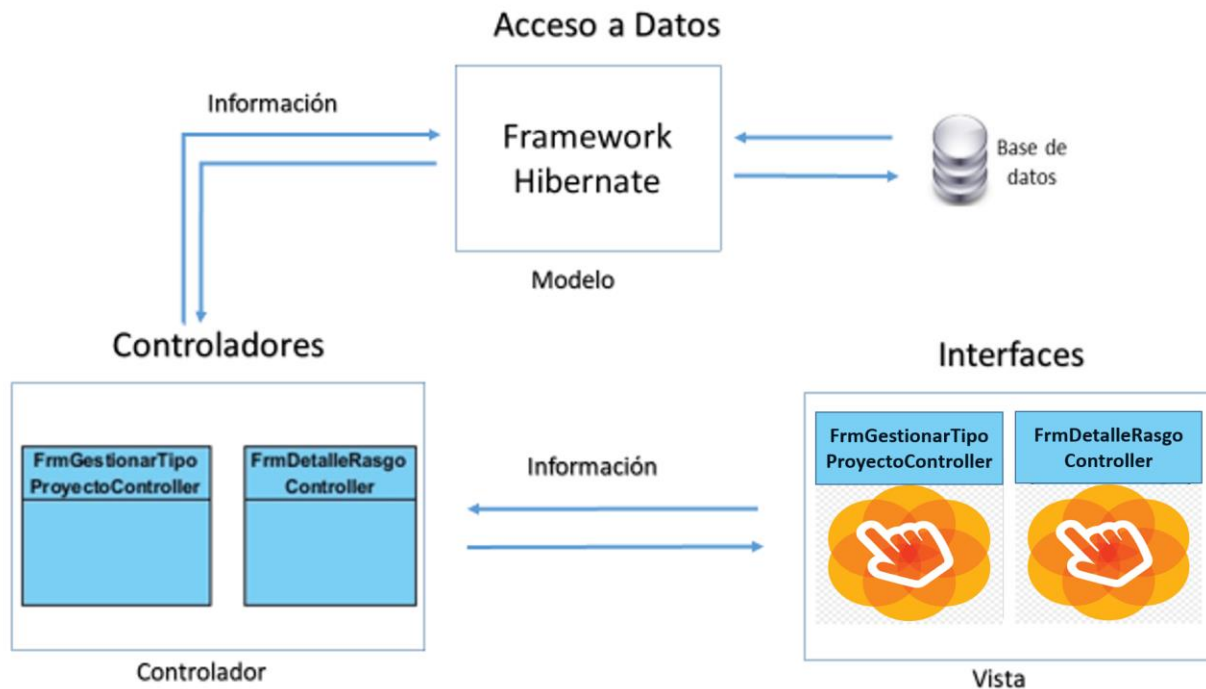


Figura 2-5 Ejemplo de la comunicación entre las capas del patrón MVC en la solución. Fuente: Elaboración propia

Con la guía de la arquitectura, se diseña el diagrama de paquete y diagrama de clase. Estos se encuentran representados en las Figuras 2-6 y 2-7 respectivamente.

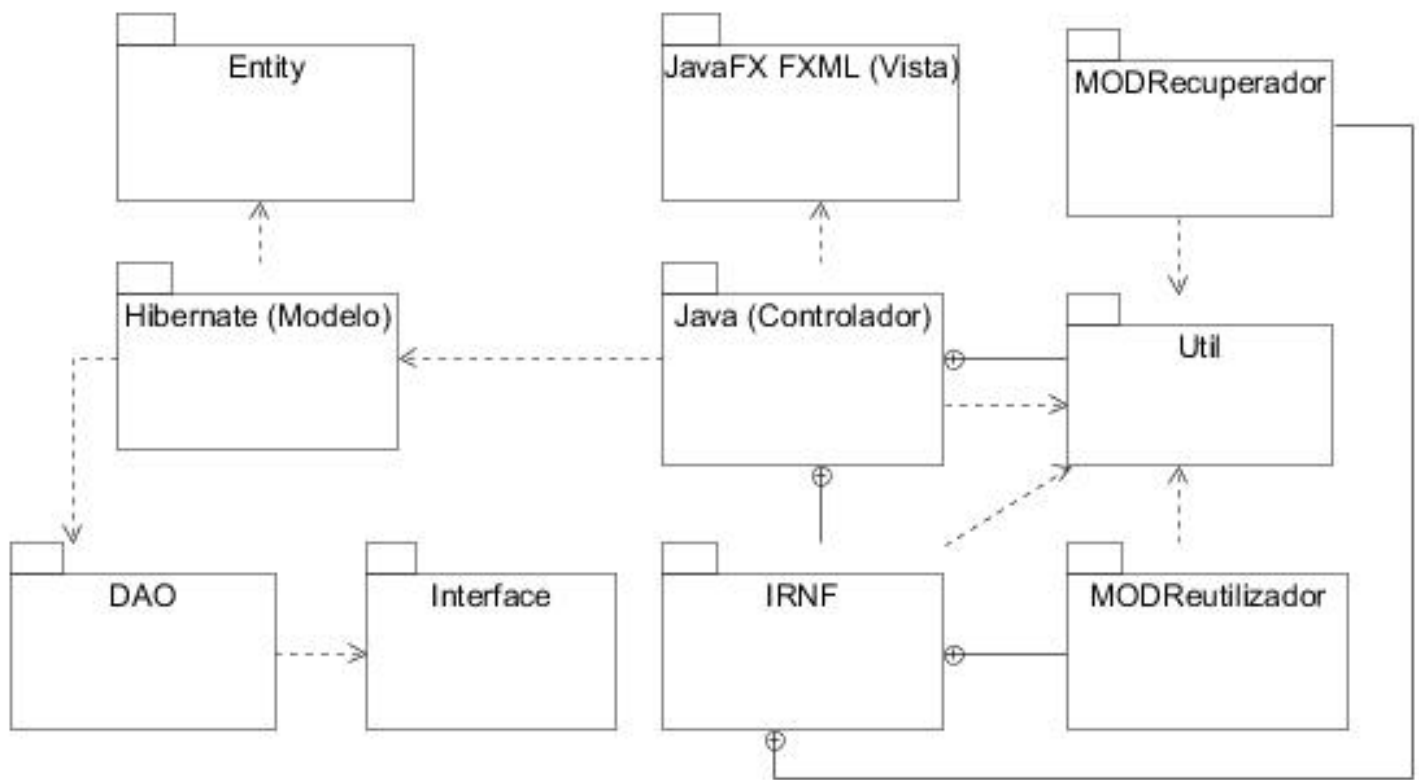


Figura 2-6. Diagrama de paquetes. Fuente: Elaboración propia

En el paquete IRNF contiene los paquetes de los módulos Recuperador y Reutilizador y la clase del algoritmo de elicitación de los RNF. Es en cada uno de estos módulos donde están las clases encargadas de ejecutar los pasos de los procesos del método de reutilización SIREN.

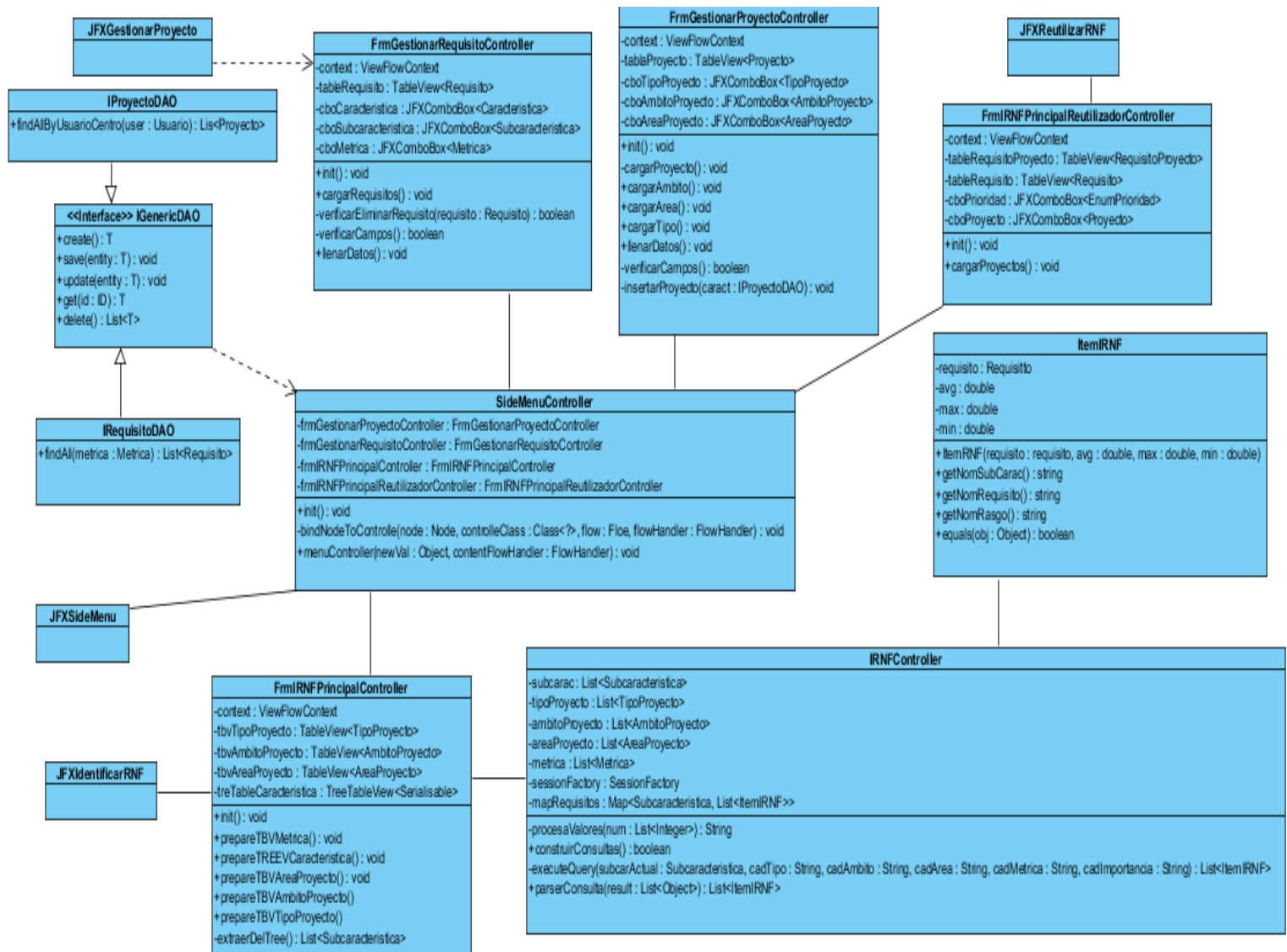


Figura 2-7. Diagrama de clases de la solución. Fuente: Elaboración propia

### 2.3.1 Modelo de datos

Para darle solución a la problemática planteada se obtuvo un modelo relacional de datos, que se generó en la herramienta CASE Visual Paradigm 8.0, para el diseño y análisis lógico de los datos. En la Figura 2-8 se representa este modelo:

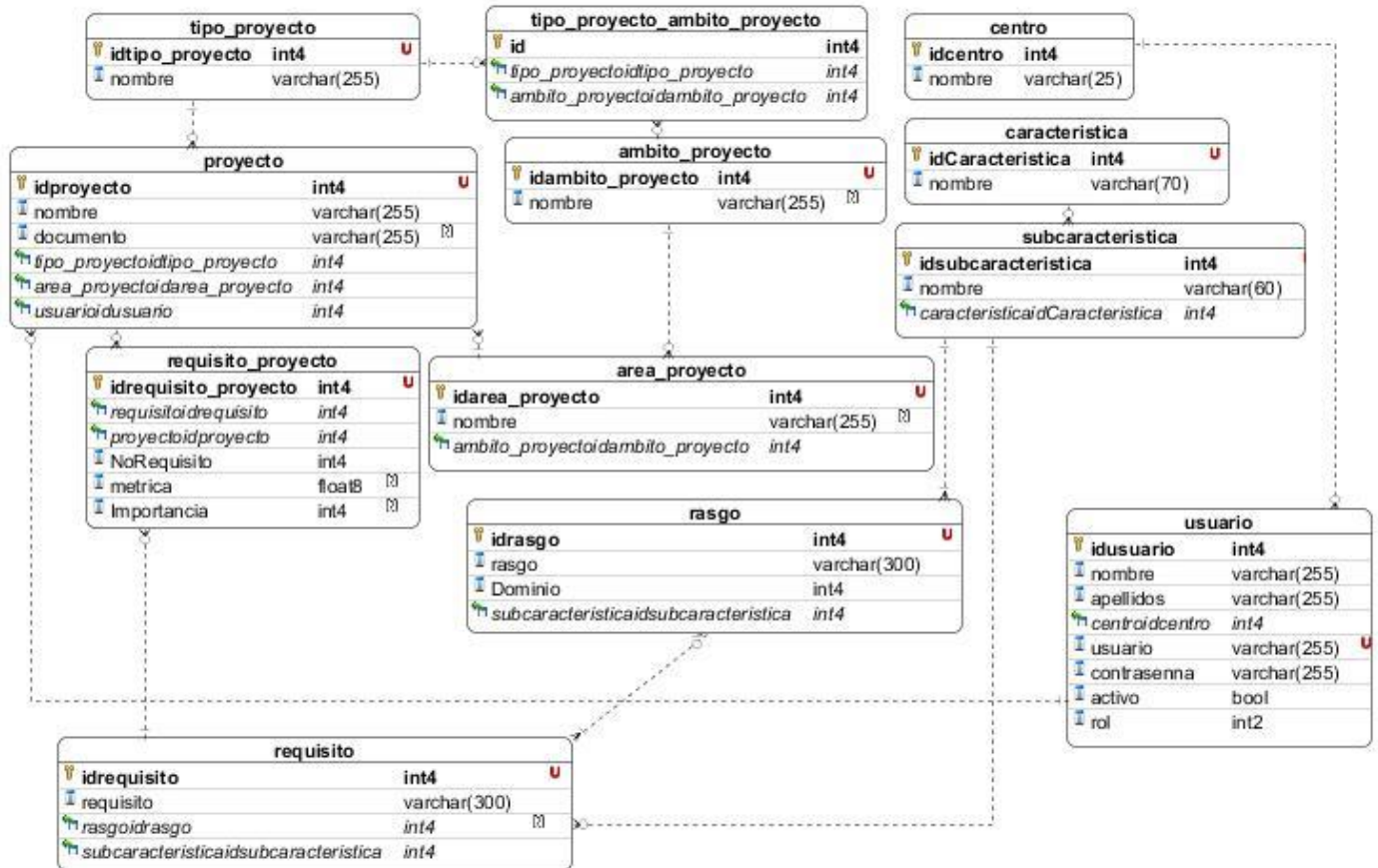


Figura 2-8. Modelo de datos de la solución. Fuente: Elaboración propia

### 2.3.2 Patrones de diseño utilizados

“Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.” En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares (Visconti y Astudillo 2015); ayudan al éxito del proyecto, pues permiten la reutilización de código, garantizan la robustez y extensibilidad del software

Lo esencial de un diseño de objetos lo constituye el diseño de las interacciones de objetos y la asignación de responsabilidades. Las decisiones que se tomen pueden influir profundamente en la extensibilidad, claridad y mantenimiento del sistema de software de objetos, además en el grado y calidad de los



componentes reutilizables, por esta razón, durante el diseño se deben realizar los casos de usos con objetos basado en los patrones GRASP (Giraldo, Acevedo y Moreno 2011).

Los patrones GRASP codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Los patrones utilizados en el desarrollo de la investigación son (Giraldo, Acevedo y Moreno 2011)

En la modelación de la solución se emplean los siguientes:

- Patrón controlador: Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades como validaciones y seguridad. El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Ejemplo de ello se tiene al hacer los cambios de interfaces, el flujo entre ellas es responsabilidad de la librería *IFlow*.

```
@FXML
private void btnAceptar_OnClick(ActionEvent event) throws IOException, FlowException {
    IRNFController controlIRNF = new IRNFController(tbvTipoProyecto.getItems(), tbvAmbitoProyecto.getItems(),
        tbvAreaProyecto.getItems(), extraerDelTree(), extraerDelTreeMetrica());
    if (!controlIRNF.construirConsultas()) {
        MensajeDeValidacion(txtBuscarAmbitoProyecto, txtBuscarAmbitoProyecto,
            "Información", "Rellene todos los campos obligatorios");
        return;
    }
    ViewFlowContext flowContext = new ViewFlowContext();
    flowContext.register("user", user);
    Flow flow = new Flow(FrmIRNFPrincipalReutilizadorController.class);
    DefaultFlowContainer container = new DefaultFlowContainer();
    FlowHandler handler = flow.createHandler(flowContext);
    Tab recup = handler.startInTab(container);
    handler.getCurrentViewMetadata().get().setTitle("Reutilizar RNF");
    ((FrmIRNFPrincipalReutilizadorController) (handler.getCurrentView().getViewContext().
        getController())).setMapRequisitos(controlIRNF.getMapRequisitos());
    recup.setClosable(true);
    if (tabPanePrincipal.getTabs().size() > 1) {
        tabPanePrincipal.getTabs().remove(tabPanePrincipal.getTabs().size() - 1);
    }
    tabPanePrincipal.getTabs().add(recup);
    tabPanePrincipal.getSelectionModel().selectLast();
}
```

Figura 2-9. Ejemplo del uso del patrón controlador utilizando *IFlow*. Fuente: Elaboración propia

- Patrón creador: El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos. Se emplea al momento de definirlos, se garantiza que si una clase puede crear objetos de otra es porque guardan determinada relación, ya sea que una contiene a la otra, o que una contiene los datos que la otra necesita. Ejemplo de ello se tiene al hacer la asignación de los controles para cargar cada una de las interfaces, la asignación es responsabilidad de la librería *Flow*.

```

@PostConstruct
public void init() {
    Objects.requireNonNull(context, "context");
    FlowHandler contentFlowHandler = (FlowHandler) context.getRegisteredObject("ContentFlowHandler");
    sideList.propagateMouseEventsToParent();
    sideList.getSelectionModel().selectedItemProperty().addListener((o, oldVal, newVal) -> {
        menuController(newVal, contentFlowHandler);
    });
    Flow contentFlow = (Flow) context.getRegisteredObject("ContentFlow");
    bindNodeToController(lblListarCaracteristica, FrmGestionarCaracteristicaController.class,
        contentFlow, contentFlowHandler);
    bindNodeToController(lblListarProyecto, FrmGestionarProyectoController.class, contentFlow,
        contentFlowHandler);
    bindNodeToController(lblListarRequisitos, FrmGestionarRequisitoController.class, contentFlow,
        contentFlowHandler);
    bindNodeToController(lblListarMetrica, FrmGestionarMetricaController.class, contentFlow,
        contentFlowHandler);
    bindNodeToController(lblListarAmbito, FrmGestionarAmbitoProyectoController.class, contentFlow,
        contentFlowHandler);
    bindNodeToController(lblListarTipoo, FrmGestionarTipoProyectoController.class, contentFlow,
        contentFlowHandler);
    bindNodeToController(lblIRNF, FrmIRNFPrincipalController.class, contentFlow, contentFlowHandler);
    bindNodeToController(lblInicio, FrmInicioController.class, contentFlow, contentFlowHandler);
}

private void bindNodeToController(Node node, Class<?> controllerClass, Flow flow, FlowHandler flowHandler) {
    flow.withGlobalLink(node.getId(), controllerClass);
}

```

Figura 2-10. Ejemplo del uso del patrón creador utilizando *Flow*. Fuente: Elaboración propia

- Patrón alta cohesión: Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable, lo que facilita la organización del proyecto y que estas no estén sobrecargadas de funcionalidades ajenas. Esto se logra en cada clase del diseño y se ejemplifica con la clase controladora que es la encargada de realizar la autenticación en el sistema.

```

private boolean verificarLogin(){
    if( username.getText() == null || password.getText() == null )return false;
    if( username.getText().trim().isEmpty() || password.getText().trim().isEmpty() )return false;

    IUsuarioDAO control = new UsuarioDAOImplHibernate();
    user = null;
    try {
        user = control.login(username.getText(), password.getText());
    } catch (mHibernateException ex) {
        Logger.getLogger(FrmLoginController.class.getName()).log(Level.SEVERE, null, ex);
    }
    return user != null;
}

void loadMain() throws FlowException {
    Stage stage = new Stage();
    Flow flow = new Flow(FrmPrincipalController.class);
    DefaultFlowContainer container = new DefaultFlowContainer();
    flowContext = new ViewFlowContext();
    flowContext.register("Stage", stage);
    flowContext.register("user", user);
    flow.createHandler(flowContext).start(container);
    JFXDecorator decorator = new JFXDecorator(stage, container.getView());
    decorator.setCustomMaximize(true);
    decorator.setGraphic(new SVGGlyph(""));
    stage.setTitle("Identificador RNF");
    double width = 800;
    double height = 600;
    Scene scene = new Scene(decorator, width, height);
    final ObservableList<String> stylesheets = scene.getStylesheets();
    stylesheets.addAll(tesis.class.getResource("/css/jfoenix-fonts.css").toExternalForm(),
        tesis.class.getResource("/css/jfoenix-design.css").toExternalForm(),
        tesis.class.getResource("/vista/css/jfoenix-main-demo.css").toExternalForm());
    stage.setScene(scene);
    stage.show();
}

```

Figura 2-11. Ejemplo del uso del patrón alta cohesión. Fuente: Elaboración propia

- Patrón experto: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase “sencillas” y más cohesivas que son más fáciles de comprender y de mantener. Esto se logra en cada clase de acceso a datos y se ejemplifica con la clase genérica GenericDAOController, que es la encargada de realizar las funciones de insertar, modificar, eliminar y obtener datos del modelo de datos.

```

public GenericDAOImplHibernate() {
    sessionFactory = HibernateUtil.getSessionFactory();
}

@Override
public T create() throws mHibernateException {
    try {
        return getEntityClass().newInstance();
    } catch (InstantiationException | IllegalAccessException ex) {
        throw new RuntimeException(ex);
    } catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}

@Override
public void update(T entity) throws mHibernateException {
    Session session = sessionFactory.openSession();
    try {
        session.beginTransaction();
        session.update(entity);
        session.getTransaction().commit();
    }
    catch (Exception ex) {
        try {
            if (session.getTransaction().isActive()) {
                session.getTransaction().rollback();
            }
        } catch (Exception exc) {
            LOGGER.log(Level.WARNING, "Falló al hacer un rollback", exc);
        }
        throw new RuntimeException(ex);
    } finally {
        session.close();
    }
}
}

```

Figura 2-12. Ejemplo del uso del patrón experto. Fuente: Elaboración propia

## 2.4 Conclusiones del capítulo

En este capítulo se describe utilizando la metodología AUP-UCI en su escenario 2, la solución generando la documentación de la herramienta para la elicitación de RNF. A partir de su elaboración se arriba a las siguientes conclusiones:

- La descripción de los pasos por cada uno de los procesos del método SIREN, estructuran la modelación de los conceptos que apoya la modelación del negocio.
- Son necesarios implementar 31 funcionalidades junto con RNF de usabilidad, software y hardware para que la herramienta apoye el trabajo de los analistas de sistemas en el proceso de elicitación de requisitos.
- Se utiliza una arquitectura en capas utilizando el patrón arquitectónico modelo-vista-controlador para independizar cada uno de estos elementos del resto de forma que no interfieran entre ellas en la implementación y mantenimiento de cada una.

## Capítulo 3. Implementación y pruebas de la herramienta para la identificación de requisitos no funcionales de software

Después de haber analizado y diseñado la solución se procede al desarrollo de la aplicación a través de la disciplina Implementación.

### 3.1 Implementación

La documentación de esta disciplina se basa fundamentalmente en el código fuente de la aplicación, pero se emplean buenas prácticas para su escritura. Cada programador tiene su propia forma de escribir los códigos y puede ser completamente diferente a la de otros programadores, pero de la forma que se use dependerá la facilidad con que otros programadores entiendan el código y se les facilite su reutilización. De ahí se desprende la importancia de los estilos de programación; conocidos como estándares de código los cuales definen un grupo de convenciones para escribir código fuente en ciertos lenguajes de programación. (Amparo López Gaona 2019) A continuación, se define la relación de estándares de codificación a utilizar en la implementación del sistema.

Identificadores: Si se trata del nombre de una clase, interfaz o enumerador se escribirá comenzado con mayúscula. Cualquier otro identificador iniciará con minúscula. Los nombres de los métodos, por indicar acciones, comenzarán con un infinitivo y los atributos serán sustantivos al describir una característica de la clase. A modo de ejemplificación y tomando como base el fragmento de código de la Figura 3-1, se visualiza como el nombre de la clase es `GeneratePDFFileIText`, como nombre del método `ConstruirPDF` y como atributo `proySeleccionado`

```

private void btnExportar_OnClick(ActionEvent event) {
    if (proySeleccionado == null) {
        return;
    }

    FileChooser save = new FileChooser();
    FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("Archivos PDF", "*.pdf");
    save.getExtensionFilters().add(extFilter);
    File file_directory = save.showSaveDialog(this.stageactual);
    if (file_directory != null) {
        //File file_directory = save.getSelectedFile();
        GeneratePDFFileIText gen = new GeneratePDFFileIText();
        try {
            gen.ConstruirPDF(new FileOutputStream(file_directory + ".pdf"), proySeleccionado);
        } catch (DocumentException | IOException | SQLException e) {

            Logger.getLogger(FrmGestionarProyectoController.class.getName()).log(Level.SEVERE, null, e);
        }

        //Abrir el PDF creado
        try {
            //Para Windows y Linux
            File path = new File(file_directory + ".pdf");
            Desktop.getDesktop().open(path);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Figura 3-1. Ejemplo visualizar la aplicación de estándar de código para identificadores. Fuente: Elaboración propia

Máxima longitud de líneas: Todas las líneas deben estar limitadas a un máximo de cien caracteres.

Espacios en blanco en expresiones y sentencias: Deben rodearse con exactamente un espacio los siguientes operadores binarios:

- Asignación (=)
- Asignación de aumento (+=, -=, etcétera)
- Comparación (==, <, >, >=, <=, !=, <>, in, not in, is, is not)
- Expresiones lógicas (and, or, not)

Ver su uso en la Figura 3-2.

```

if( username.getText() == null || password.getText() == null )return false;
if( username.getText().trim().isEmpty() || password.getText().trim().isEmpty() )return false;

IUsuarioDAO control = new UsuarioDAOImplHibernate();
user = null;
try {
    user = control.login(username.getText(), password.getText());
} catch (mHibernateException ex) {
    Logger.getLogger(FrmLoginController.class.getName()).log(Level.SEVERE, null, ex);
}
return user != null;

```

Figura 3-2. Ejemplo visualizar la aplicación de estándar de código para espacio en expresiones y sentencias. Fuente: Elaboración propia

### Importaciones:

- Las importaciones deben estar en líneas separadas.
- Siempre deben colocarse al comienzo del archivo.
- Deben quedar agrupadas de la siguiente forma:
  - ✓ Importaciones de la librería estándar.
  - ✓ Importaciones terceras relacionadas.
  - ✓ Importaciones locales de la aplicación/librerías.
- Cada grupo de importaciones debe estar separado por una línea en blanco.

Un ejemplo de código que muestra la aplicación de este estándar de código se muestra en la figura 3-3.

```

package controlador;

import java.net.URL;
import java.util.ResourceBundle;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Scene;
import javafx.stage.Stage;

import javax.annotation.PostConstruct;
import com.jfoenix.controls.JFXDecorator;
import com.jfoenix.controls.JFXPasswordField;
import com.jfoenix.controls.JFXTextField;
import com.jfoenix.svg.SVGGlyph;
import io.datafx.controller.ViewController;
import io.datafx.controller.flow.Flow;
import io.datafx.controller.flow.FlowException;
import io.datafx.controller.flow.container.DefaultFlowContainer;
import io.datafx.controller.flow.context.FXMLViewFlowContext;
import io.datafx.controller.flow.context.ViewFlowContext;

import modelo.DAO.Interfaces.IUsuarioDAO;
import modelo.DAO.UsuarioDAOImplHibernate;
import modelo.Entity.Usuario;
import modelo.exception.mHibernateException;

```

Figura 3-3. Fragmento de código que representa el uso del estándar de codificación Importaciones. Fuente:  
Elaboración propia

### Comentarios:

- Los comentarios deben ser oraciones completas.
- Si un comentario es una frase u oración, su primera palabra debe comenzar con mayúscula a menos que sea un identificador que comience con minúscula.
- Si un comentario es corto, el punto final puede omitirse.



```

/**
 *
 * @param nombreI: Nombre de la Interfaz que se va a cargar
 * @param dat: Instancia de DB
 * @param Sactual: Stage que la crea
 * @param datos: Datos opcionales que se deseen pasar, ejemplo a la hora de
 * modificar un usuario
 * @param cerrar: true si se desea cerrar el Stage quién lo crea, false en
 * caso contrario
 * @return el controlador del Stage que se crea
 * @throws IOException
 */

```

Figura 3-4. Fragmento de código que muestra el uso de comentarios. Fuente: Elaboración propia

### Variables y constantes:

- Los nombres de las variables siempre comenzarán con minúscula.
- Los nombres de las constantes siempre se escribirán en mayúsculas

```

public static String NIInicio = "FrmInicio";
public static String NIPrincipal = "FrmPrincipal";
public static String NIGestionarRequisitoProyecto = "FrmGestionarRequisitoProyecto";
public static String NIGestionarRequisito = "FrmGestionarRequisito";
public static String NIGestionarRasgo = "FrmGestionarRasgo";
public static String NIGestionarProyecto = "FrmGestionarProyecto";
public static String NIGestionarCaracteristica = "FrmGestionarCaracteristica";
public static String NIDetalleRequisitoProyecto = "FrmDetalleRequisitoProyecto";
public static String NIGestionarTipoProyecto = "FrmGestionarTipoProyecto";
public static String NIGestionarCentro = "FrmGestionarCentro";
public static String NIGestionarUsuario = "FrmGestionarUsuario";
public static String NISBCPrincipal = "FrmSBCPrincipal";
public static String NISBCDetalleRasgoProyecto = "FrmSBCDetalleRasgoProyecto";
public static String NISBCAvanzado = "FrmSBCAvanzado";
public static String NISBCPrincipalReutilizador = "FrmSBCPrincipalReutilizador";
public static String NIDetalleRasgo = "FrmDetalleRasgo";
public static String NIDetalleRequisito = "FrmDetalleRequisito";
public static String NIDetalleProyecto = "FrmDetalleProyecto";
public static String NIDetalleTipoProyecto = "FrmDetalleTipoProyecto";
public static String NIConfiguracionesProyecto = "FrmConfiguracionesProyecto";
public static String NIDetalleAmbitoProyecto = "FrmDetalleAmbitoProyecto";
public static String NIGestionarAmbitoProyecto = "FrmGestionarAmbitoProyecto";
public static String NIDetalleAreaProyecto = "FrmDetalleAreaProyecto";

public static String NIIRNFPrincipalRecuperador = "FrmIRNFPrincipal";
public static String NIIRNFPrincipalReutilizador = "FrmIRNFPrincipalReutilizador";
public static String NIDetalleIRNF = "FrmDetalleIRNF";

```

Figura 3-5 Fragmento de código que muestra el uso de variables y constantes. Fuente: Elaboración propia

### Alineación y espacios en blanco:

- Separar las funciones de alto nivel y definiciones de clases con dos líneas en blanco.
- Las definiciones de métodos dentro de una clase deben separarse por una línea en blanco.
- Todos los bloques deben estar alineados de tal manera que sean fácilmente distinguibles.
- Dentro de un bloque todas las instrucciones van a la misma estructura.

En los fragmentos de código mostrados en las Figuras 3-1 y 3-5 se puede apreciar su resultado en su conjunto y cómo queda organizado el código.

```
for (Metrica item : rasgo) {
    if (item.getSubcaracteristica() == null) {
        continue;
    }
    if (item.getSubcaracteristica().equals(subca)) {
        if (item.getSelectedProperty().get()) {
            rasgosActual.add(item);
        }
    }
}
list.clear();
rasgosActual.stream().forEach((item) -> {
    list.add(item.getIdmetrica());
});
if (sinmet)list.add(SNMetrica.getIdmetrica());

String cadrasgo = procesaValores(list);
if (cadrasgo.isEmpty()) {
    continue;
}
list.clear();
list.add(subca.getImportancia().getImportancia());
String cadImportancia = procesaValores(list);
if (cadImportancia.isEmpty()) {
    return false;
}
if (!mapRequisitos.containsKey(subca)) {
    mapRequisitos.put(subca, executeQuery(subca, cadTipoProyecto, cadAmbitoProyecto, cadAreaProyecto
}
}
```

Figura 3-6. Fragmento de código que muestra el uso de alineación y espacios en blanco. Fuente: Elaboración propia

#### **3.1.1 Despliegue de la herramienta**

El diagrama de despliegue es un plano que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. Permite modelar la disposición física o topología de un sistema, muestra el hardware y los componentes instalados en el hardware y permite mostrar las conexiones físicas entre el hardware y las relaciones entre componentes (Hernandez 2013). A continuación, se muestra en la Figura 3-6 el diagrama de despliegue, donde se visualiza la distribución de los componentes de software en los nodos físicos.

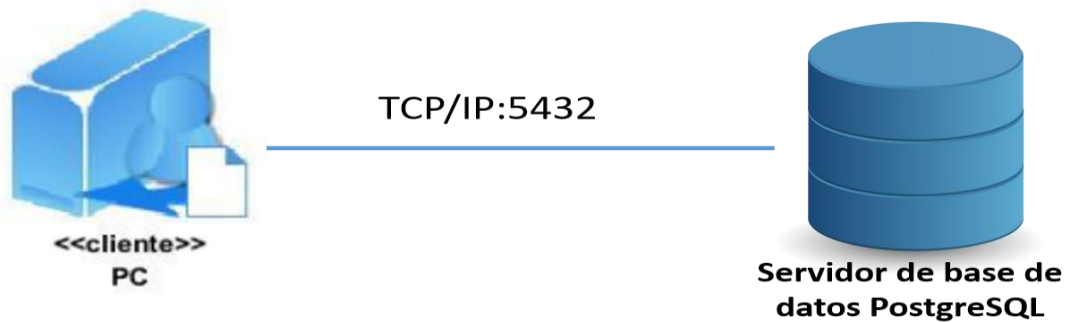


Figura 3-7. Diagrama de despliegue. Fuente: Elaboración propia

En el nodo PC que representa la computadora que utiliza el analista y/o administrador para interactuar con la aplicación que debe tener instaladas la máquina virtual de Java a partir de la versión 8.0.91 y la herramienta. Esta computadora se comunica con el servidor de base de datos PostgreSQL donde se almacenan los datos de la herramienta, mediante el protocolo TCP/IP:5432. El servidor debe tener instalado PostgreSQL a partir de su versión 9.3.

Luego de implementado el sistema, se procede a ejecutar la fase Pruebas de la metodología AUP-UCI. De esta fase solo se emplearán las disciplinas pruebas internas y pruebas de aceptación puesto que no aplica a la investigación, las pruebas de liberación.

### 3.2 Pruebas internas

Durante esta etapa se prueban los componentes del producto con el objetivo de medir la calidad del software. El proceso de pruebas está encaminado a medir el cumplimiento de las funcionalidades establecidas por el cliente, reduciendo de esta manera el número de errores no detectados.

Entre las pruebas internas se decide hacer pruebas unitarias, de caja blanca y de camino básico. También se emplea la herramienta JUnit que permite correr un conjunto de pruebas de forma automática e informa de aquellas que han fallado (Montes Ramírez et al. 2016).

Las pruebas unitarias o prueba de unidad enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software. Al usar la descripción del diseño de componente como guía, las rutas de control importantes se prueban para descubrir errores dentro de la frontera del módulo. La relativa complejidad de las pruebas y los errores que descubren están limitados por el ámbito restringido que se establece para la prueba de unidad. Las pruebas de unidad se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un componente. Este tipo de pruebas puede realizarse en paralelo para múltiples componentes. (Sommerville 2015)

Las pruebas realizadas mediante el método de caja blanca se basan en el examen cercano de los detalles de procedimiento. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles. En ocasiones llamada prueba de caja de vidrio, es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. (Pressman 2010)

La prueba de camino básico permite 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. (Sommerville 2015)

La complejidad ciclomática se aplica a todos los métodos de la herramienta, pero en la investigación se analizará la funcionalidad de identificación de requisitos no funcionales. Al calcular la complejidad ciclomática a partir del grafo realizado del método en cuestión, se llegó a la conclusión de que se necesitan 8 casos de prueba para validar todas las posibles variantes del método. El algoritmo de identificación de requisitos posee poco riesgo debido a que el resultado arrojado por la métrica pertenece al intervalo entre 1 - 8.

Después de aplicar esta complejidad ciclomática se procede a la realización de la técnica de prueba automatizada, pues esto facilita identificar funciones que no ofrecen una salida acorde con la lógica que se deseaba implementar. A continuación, se representa una de las funcionalidades del sistema a la cual se le realiza una prueba unitaria:

```

54  @Test
55  public void testConstruirConsultas() {
56      // TODO review the generated test code and remove the default call to fail.
57      AmbitoProyectoDAOImplHibernate cambito= new AmbitoProyectoDAOImplHibernate();
58      AreaProyectoDAOImplHibernate carea= new AreaProyectoDAOImplHibernate();
59      TipoProyectoDAOImplHibernate cTipo = new TipoProyectoDAOImplHibernate();
60      SubcaracteristicaDAOImplHibernate cSubca = new SubcaracteristicaDAOImplHibernate();
61      MetricaDAOImplHibernate crasgo = new MetricaDAOImplHibernate();
62      try {
63          List<TipoProyecto> lTipo = cTipo.findAll();
64          List<AmbitoProyecto>lAmbito = cambito.findAll();
65          List<AreaProyecto>lArea = carea.findAll();
66          List<Subcaracteristica>lSubca = cSubca.findAll();
67          List<Metrica>lmtrica = crasgo.findAll();
68          IRNFController control = new IRNFController(lTipo, lAmbito, lArea, lSubca,lmtrica);
69          control.construirConsultas();
70
71      } catch (mHibernateException ex) {
72          Logger.getLogger(IRNFControllerTest.class.getName()).log(Level.SEVERE, null, ex);
73          fail("Ha fallado la función");
74      }
75  }
76  }
77  }
78  }
79  }

```

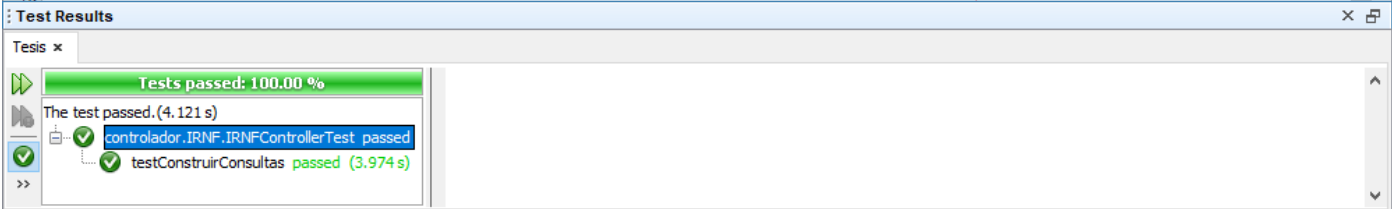


Figura 3-8. Resultados de la prueba realizada en JUnit. Fuente: Elaboración propia

En el caso de la prueba que se muestra en la Figura 3-7, no se detectaron errores. La prueba se realiza en 4.121 segundos y el método testConstruirConsultas 3.974 segundos.

Una vez realizada la corrección de los errores detectados, los métodos de cada clase se ejecutaron de forma correcta, obteniendo de ellos los resultados esperados. Se tomaron precauciones para evitar errores similares y se definieron nuevos casos de prueba para verificar el correcto desempeño de las funcionalidades.

También se emplearon pruebas de caja negra para probar las funcionalidades tomando como unidad a cada interfaz. Aquí no se centra en cómo se generan las respuestas del sistema, solo se analizan los datos de entrada y los resultados obtenidos.

Este método contiene la técnica de partición de equivalencia que consiste en clasificar las entradas de datos del sistema en grupos que presentan un comportamiento similar, por lo cual serán procesados de la misma forma. Se pueden definir particiones tanto para valores válidos como inválidos.

A continuación, se muestran los casos de pruebas de caja negra aplicado a los CU:

Tabla 3-1. Caso de prueba de caja negra "Gestionar repositorio de requisitos". Fuente: Elaboración propia

Escenario	Variables				Descripción	Respuesta del sistema	Flujo Central
	1	2	3	4			
<b>Sección 1: Registrar requisito</b>							
EC 1.1 Registrar requisito satisfactoriamente	V	V	V	V	El analista introduce todos los datos correctamente, permitiendo el sistema la adición de un nuevo requisito.	Se muestra un mensaje de información "El requisito ha sido creado correctamente".	1.- Introducir el requisito, seleccionar la característica, subcaracterística y la métrica. 2.- Presionar el botón "Aceptar".
EC 1.2 Datos incorrectos.	I	I	I	I	El analista introduce uno o varios datos incorrectos, el sistema no permitirá adición de nuevo requisito.	Muestra un mensaje de error: "Los datos introducidos son incorrectos".	1.- Se introduce uno o varios datos incorrectos. 2.- Presionar el botón "Aceptar".
	V	I	I	I			
	I	V	I	I			
	V	V	I	I			
	I	I	V	I			
	V	I	V	I			
	I	V	V	I			
	V	V	V	I			
	I	I	I	V			
	V	I	I	V			
	I	V	I	V			
	V	V	I	V			

	I	I	V	V			
	V	I	V	V			
	I	V	V	V			
	V	V	V	V			
EC 1.3 Campos vacíos.	I	I	I	I	El analista deja uno o varios campos vacíos, el sistema no permitirá adicionar el nuevo requisito.	Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".	1.- Se dejan uno o varios campos vacíos. 2.- Presionar el botón "Aceptar".
	V	I	I	I			
	I	V	I	I			
	V	V	I	I			
	I	I	V	I			
	V	I	V	I			
	I	V	V	I			
	V	V	V	I			
	I	I	I	V			
	V	I	I	V			
	I	V	I	V			
	V	V	I	V			
	I	I	V	V			
	V	I	V	V			
	I	V	V	V			
V	V	V	V				

**Sección 2: "Modificar requisito"**

EC 2.1 Modificar requisito satisfactoriamente	V	V	V	V	El analista introduce todos los datos correctamente, permitiendo al sistema modificar el requisito.	Se muestra un mensaje de información "El requisito se ha actualizado correctamente".	1.-Introducir el requisito, seleccionar la característica, subcaracterística y la métrica. 2.- Presionar el botón "Aceptar".
EC 2.2 Datos incorrectos.	I	I	I	I	El analista introduce uno o varios datos incorrectos, el sistema no permitirá modificar el requisito.	Muestra un mensaje de error: "Los datos introducidos son incorrectos".	1.- Se introduce uno o varios datos incorrectos. 2.- Presionar el botón "Aceptar".
	V	I	I	I			
	I	V	I	I			
	V	V	I	I			
	I	I	V	I			
	V	I	V	I			
	I	V	V	I			
	V	V	V	I			
	I	I	I	V			
	V	I	I	V			
	I	V	I	V			
	V	V	I	V			
	I	I	V	V			
	V	I	V	V			
	I	V	V	V			
	V	V	V	V			



EC 2.3 Campos vacíos.	I	I	I	I	El analista deja uno o varios campos vacíos, el sistema no permitirá modificar el requisito.	Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".	1.- Se dejan uno o varios campos vacíos. 2.- Presionar el botón "Aceptar".
	V	I	I	I			
	I	V	I	I			
	V	V	I	I			
	I	I	V	I			
	V	I	V	I			
	I	V	V	I			
	V	V	V	I			
	I	I	I	V			
	V	I	I	V			
	I	V	I	V			
	V	V	I	V			
	I	I	V	V			
	V	I	V	V			
	I	V	V	V			
V	V	V	V				
<b>Sección 3: "Eliminar requisito"</b>							
EC 3.1 Eliminar el requisito satisfactoriamente.	El analista selecciona el requisito que desea eliminar.				Muestra un mensaje de confirmación "¿Seguro que desea eliminar el requisito seleccionado?"	1.- Presiona el botón "Aceptar".	

Tabla 3-2. Caso de prueba de caja negra "Identificar RNF". Fuente: Elaboración propia

Escenario	Variables					Descripción	Respuesta del sistema	Flujo Central
	1	2	3	4	5			
<b>Sección 1: Identificar RNF</b>								
EC 1.1 Identificar requisitos no funcionales satisfactoriamente	V	V	V	V	V	El analista introduce todos los datos correctamente, permitiendo al sistema identificar los requisitos no funcionales.	Se muestra la interfaz Reutilizar RNF.	1.-Introducir el tipo, ámbito y área del proyecto, subcaracterísticas y las métricas. 2.- Presionar el botón "Aceptar".
EC 1.2 Seleccionar al menos un dato de cada característica.	I	I	I	I	I	El analista no selecciona uno o varias características el identificar los requisitos no funcionales.	Muestra un mensaje de error: "Existen campos obligatorios vacíos. Por favor, complete estos campos".	1.-No introducir el tipo, ámbito o área del proyecto, subcaracterísticas o las métricas. 2.- Presionar el botón "Aceptar".
	V	I	I	I	I			
	I	V	I	I	I			
	V	V	I	I	I			
	I	I	V	I	I			
	V	I	V	I	I			
	I	V	V	I	I			
	V	V	V	I	I			
	I	I	I	V	I			
	V	I	I	V	I			
	I	V	I	V	I			
	V	V	I	V	I			

	I	I	V	V	I
	V	I	V	V	I
	I	V	V	V	I
	V	V	V	V	I
	I	I	I	I	V
	V	I	I	I	V
	I	V	I	I	V
	V	V	I	I	V
	I	I	V	I	V
	V	I	V	I	V
	I	V	V	I	V
	V	V	V	I	V
	I	I	I	V	V
	V	I	I	V	V
	I	V	I	V	V
	V	V	I	V	V
	I	I	V	V	V
	V	I	V	V	V
	I	V	V	V	V
	V	V	V	V	V

Este proceso de prueba permitió verificar el cumplimiento de los requisitos funcionales del software, donde los resultados de las pruebas que no fueron satisfactorios pasaron a ser no conformidades.

En la Figura 3-9 se muestra las no conformidades que se identificaron en cada iteración, asociadas a cada caso de uso. En la primera iteración se identificaron 11 No Conformidades (NC), de ellas 9 fueron funcionales, una de validación y una de ortografía. En la segunda iteración 6 NC, de ellas 5 funcionales y una de validación. En la tercera iteración no se encontraron NC, quedando demostrado que los errores encontrados fueron resueltos en su totalidad, para un total de 17 NC. A continuación, se muestran las no conformidades que se identificaron en cada iteración:

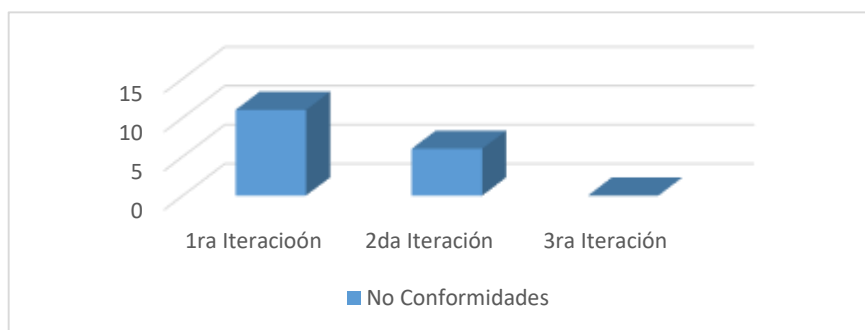


Figura 3-9. No conformidades encontradas en cada iteración de la técnica partición de equivalencia. Fuente: Elaboración propia

### 3.3 Pruebas de aceptación

Para realizar la prueba de aceptación se empleó la técnica de prueba alfa, por el cliente. Se evaluaron las funcionalidades de la herramienta de identificación de requisitos no funcionales de software.

Las pruebas alfa se realizan en la primera versión del programa, la cual es enviada a los verificadores para probarla. Algunos equipos de desarrollo utilizan el término alfa informalmente para referirse a una fase donde un producto todavía es inestable, aguarda todavía a que se eliminen los errores o a la puesta en práctica completa de toda su funcionalidad, pero satisface la mayoría de los requisitos. (Díaz, Macís y Solano 2017)

Con el propósito de comprobar que la herramienta desarrollada satisface las necesidades de los proyectos de la UCI, se realizan pruebas de aceptación en el proyecto XAVIA-SIDEC del centro Informática Médica (CESIM). Durante su ejecución participaron el analista principal y el jefe de proyecto los que, luego de ejecutar las funcionalidades del sistema, emitieron un conjunto de recomendaciones que fueron tenidas en

cuenta para el mejoramiento de la propuesta de solución a la problemática planteada, además emitieron su conformidad con la misma evidenciado en el acta de aceptación que se muestra en la Figura 3-8.

Entre las recomendaciones estuvo

- Emplear el marco de trabajo JFoenix para mejorar el diseño visual de la herramienta.
- Mostrar los datos de los proyectos agrupados y en la parte de debajo de la interfaz.
- Colocar el menú en la parte izquierda de la aplicación y que fuera desplegable.

La Habana, mayo 28 de 2019

Luego de haber revisado la herramienta "IRNF Software para la identificación de requisitos no funcionales", a propósito de su presentación como Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Como resultado del trabajo de investigación, desarrollo e innovación tecnológica fueron obtenidos 7 componentes de software:

- Módulo para la gestión de requisitos no funcionales.
- Módulo para la gestión de proyectos.
- Módulo para la gestión de métricas.
- Módulo para la gestión de usuarios.
- Módulo para la gestión características y subcaracterísticas de requisitos no funcionales.
- Módulo para la identificación de requisitos no funcionales.
- Módulo para la reutilización de requisitos no funcionales.

Se considera que los resultados obtenidos son de gran valor práctico y social para el desarrollo de software en la Universidad de las Ciencias Informáticas.

Luego del intercambio realizado con el autor de la propuesta, teniendo en cuenta los elementos mencionados anteriormente y en correspondencia con el valor científico, técnico y social de la misma se decide aprobar y avalar el resultado obtenido. Se considera que la investigación desarrollada reúne los requisitos para ser presentado como resultado práctico de la investigación del autor.

Firman la presente:

Nombre y Apellidos: Ing. Victor Manuel Quintana Díaz

Cargo: Jefe de proyecto



Firma



Figura 3-10. Acta de aceptación emitida por el Jefe del proyecto XAVIA-SIDEC

### 3.4 Conclusiones del capítulo

En este capítulo se obtuvo como resultado la descripción de los estándares de codificación empleados, así como la descripción de las pruebas realizadas y los principales resultados. Las conclusiones obtenidas son:

- Las pruebas realizadas validan que la herramienta funciona correctamente y lo hace bien, determinado por las pruebas realizadas tanto al código como a las funcionalidades.
- Existe satisfacción con la herramienta y esto se corrobora con la carta de aceptación obtenida luego de emplear la herramienta en el proyecto XAVIA-SIDEC .

## Conclusiones

Al término de esta investigación se afirma que los objetivos planteados en su diseño metodológico fueron cumplidos y se determina que:

- Se obtiene como resultado una fundamentación de la aplicación a partir del estudio del concepto de ingeniería de requisitos, específicamente de la etapa de elicitación, las técnicas empleadas y el método de reutilización SIREN.
- Se determinan aspectos importantes que se consideran en el desarrollo de la aplicación a partir del estudio de herramientas que existen y cuyo objetivo es el apoyo en el proceso de elicitación de requisitos.
- Es necesaria la implementación de un total de 31 requisitos funcionales para que la aplicación cumpla las necesidades por las que fue concebida. Estos rigen el desarrollo de la solución a través de la metodología AUP-UCI en su escenario 2.
- La solución es válida dictaminado por los resultados de la aplicación de pruebas unitarias y de aceptación a la herramienta y solventar las no conformidades detectadas y las recomendaciones realizadas durante el proceso..

## Recomendaciones

Para futuras investigaciones se recomienda:

- Desplegar la herramienta en todos los centros de la universidad para apoyar el trabajo de los analistas y la generación de la documentación de los requisitos no funcionales.
- Incorporar a la herramienta informática el análisis de los requisitos funcionales para ganar en completitud del proceso de elicitación.



## Referencias bibliográficas

- ARISTEGUI, J.L., 2010. Los casos de prueba en la prueba del software. *Lámpsakos*, no. 3, pp. 27-34.
- BUITRÓN, S.L., FLORES-RIOS, B.L. y PINO, F.J., 2018. Elicitación de requisitos no funcionales basada en la gestión de conocimiento de los stakeholders. *Ingeniare. Revista chilena de ingeniería*, vol. 26, no. 1, pp. 142-156.
- CÁCERES SALDAÑA, E.M., 2014. Metodología para el reúso efectivo de patrones de requisitos en la ingeniería de software.
- Caliber | Micro Focus. [en línea], 2019. [Consulta: 20 mayo 2019]. Disponible en: <https://www.microfocus.com/es-es/products/requirements-management-caliber/specs>.
- CANÓS, J.H. y LETELIER, P., 2012. Metodologías ágiles en el desarrollo de software. *Universidad Politécnica de Valencia* [en línea], [Consulta: 26 noviembre 2018]. Disponible en: <http://roa.ult.edu.cu/jspui/bitstream/123456789/476/1/TodoAgil.pdf>.
- CASTAÑEDA, B.P., 2015. Proceso metodológico para la mejora continua de la elicitación de requerimientos de software basado en área de proceso de manejo de requerimientos de CMMI Dev v1.3 [en línea]. Maestría. Argentina: Universidad Tecnológica Nacional, Facultad Regional. Disponible en: <https://docplayer.es/72495589-Tesis-de-maestria-en-ingenieria-en-sistemas-de-informacion.html>.
- DÍAZ, W.G., MACÍS, G.R. y SOLANO, F.R.S., 2017. Sistema de ventas para librería Sánchez, Siuna, 2016. *Revista Universitaria del Caribe*, vol. 19, no. 2, pp. 47-55.
- DURÁN, A.T., 2016. REM - Requisite Management. *REM - Requisite Management* [en línea]. [Consulta: 27 abril 2019]. Disponible en: [http://www.lsi.us.es/descargas/descarga\\_programas.php?id=3](http://www.lsi.us.es/descargas/descarga_programas.php?id=3).
- GAMMA, E., HELM, R., JOHNSON, R. y VLISSIDES, J., 2009. *Design Patterns\_ Elements of Reusable Object-Oriented Software*. S.l.: s.n.
- GARCÍA, M.P., IRRAZÁBAL, E., CARRASCO-VELAR, R. y COCA, Y., 2016. Importancia de los requisitos no funcionales: estudio preliminar en una Universidad de Cuba. *VII taller internacional de calidad en las tecnologías de la información y las comunicaciones calidad*.
- GIRALDO, G.L., ACEVEDO, J.F. y MORENO, D.A., 2011. Una ontología para la representación de conceptos de diseño de software. *Revista Avances en Sistemas e Informática*, vol. 8, no. 3.
- GÓMEZ, G.Y., 2012. *Base de conocimiento y motor de inferencia de un Sistema Basado en Conocimiento para la obtención de los requisitos de software en la línea temática de atención y tratamiento de las emergencias en el centro ISEC*. La Habana, Cuba: Universidad de las Ciencias Informáticas.
- HERNANDEZ, L., 2013. Modelo de Implementación. *22 de marzo del 2018* [en línea]. Disponible en: <http://ithleovi.blogspot.com/2013/06/unidad-5-modelo-deimplementacion-el.html>.
- HORSTMANN, C., 2007. *Big Java*. 3rd. S.l.: Wiley. ISBN 978-0-470-10554-2.
- HORSTMANN, C. y BUDD, T., 2009. *Big C++*. 2nd. S.l.: Wiley. ISBN 978-0-470-38328-5.

IBM-DOORS, 2015. DOORS. [en línea]. Disponible en: <http://www-01.ibm.com/software/awdtools/doors/productline/>.

IBM-RRC, 2016. Rational Requirements Composer. *Rational Requirements Composer* [en línea]. Disponible en: <http://www-142.ibm.com/software/products/es/es/rrc/>.

INSTITUTO NACIONAL DE TECNOLOGÍAS DE LA COMUNICACIÓN, 2008. *Guía avanzada de gestión de requisitos*. 2008. S.l.: s.n.

INTECO, 2008. Guía práctica de gestión de requisitos. [en línea]. [Consulta: 8 abril 2019]. Disponible en: [https://www.inteco.es/file/NRDmviQoTBl\\_jZcyjTYRlw](https://www.inteco.es/file/NRDmviQoTBl_jZcyjTYRlw).

JETBRAINS, 2017. JFoenix. [en línea]. [Consulta: 20 mayo 2019]. Disponible en: <http://www.jfoenix.com/>.

JUNIT, 2018. JUnit 5 User Guide. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <https://junit.org/junit5/docs/current/user-guide/>.

LASHERAS, J., ÁLVAREZ, J.A.T., NICOLÁS, J. y MOROS, B., 2003. Soporte Automatizado a la reutilización de requisitos. *JISBD*. S.l.: s.n., pp. 335-346.

AMPARO LÓPEZ GAONA, 2019. Guía de estilo para codificación en Java. *Guía de estilo para codificación en Java* [en línea]. [Consulta: 12 marzo 2019]. Disponible en: <http://hp.fciencias.unam.mx/~alg/normas/estilo.html>.

LOUCOPOULOS, P. y KARAKOSTAS, V., 1995. *System requirements engineering*. S.l.: McGraw-Hill, Inc. ISBN 0-07-707843-8.

MONTES RAMÍREZ, L., DOMINGO MARTÍNEZ, R., SEBASTIÁN LÓPEZ, M. y LANAU HERNÁNDEZ, P., 2016. Construyendo un paisaje. Megalitos, arte esquemático y cabañeras en el Pirineo Central. . S.l.:

NETBEANS, 2018. NetBeans IDE 8.2 Release information. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <https://netbeans.org/community/releases/82/>.

NIKUAL, M.M.U., 2011. La elicitación de requisitos en el contexto de un proyecto software. *Ingenierías USBMed*, vol. 2, no. 2, pp. 25-29.

NORMALIZACIÓN, O.I. de, 2011. *ISO-IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models*. S.l.: ISO.

Oficina Nacional de Normalización, 2016. NC ISO/IEC 25010:2016 Ingeniería de software y sistemas – Requisitos de la calidad y evaluación de software (square) – Modelos de la calidad. 1ra edición. La Habana. Mayo 2016.

ORACLE, 2016. What is JavaFX? | JavaFX 2 Tutorials and Documentation. [en línea]. [Consulta: 20 mayo 2019]. Disponible en: <https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm>.

ORACLE, O.H.C., 2018. JavaFX Scene Builder Information. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <https://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html>.

- PACHECO, C., GARCIA, I., CALVO-MANZANO, J.A. y ARCILLA, M, 2016. Reusing Functional Software Requirements in Small-sized Software Enterprises: A Model oriented to the Catalog of Requirements. *Requirements Engineering*, vol. 22, no. 2, pp. 275-287. DOI 10.1007/s00766-015-0243-1.
- PÉREZ HUEBE, M., 2005. *Ingeniería de requerimientos*.
- PGADMIN, 2018. pgAdmin - PostgreSQL Tools. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <https://www.pgadmin.org/>.
- POSTGRESQL, 2018. PostgreSQL: About. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <https://www.postgresql.org/about/>.
- PRESSMAN, R.S., 2010. *Ingeniería del software, un enfoque práctico*. 7ma edición. Madrid, España: s.n.
- SÁNCHEZ, T.R., 2015. Metodología de desarrollo para la Actividad productiva de la UCI. pp. 1-16.
- SAWYER, P. y KOTONYA, G., 2001. Software requirements. *SWEBOK*, pp. 9.
- SELLARÈS, T., 2011. The Model View Controller: a Composed Pattern. *Universitat de Girona* [en línea], Disponible en: <http://ima.udg.edu/~sellares/EINF-ES1/MVC-Toni.pdf>.
- SIEGEL, J., 2005. What is UML | Unified Modeling Language. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <http://www.uml.org/what-is-uml.htm>.
- SOMMERVILLE, I., 2015. *Software Engineering GE*. S.I.: Pearson Australia Pty Limited.
- SOMMERVILLE, I. y SAWYER, P., 1997. *Requirements engineering: a good practice guide*. S.I.: John Wiley & Sons, Inc. ISBN 0-471-97444-7.
- STANDISH GROUP, 2019. The Rise and Fall of the Chaos Report Figures. *ResearchGate* [en línea]. [Consulta: 31 mayo 2019]. Disponible en: [https://www.researchgate.net/publication/220092178\\_The\\_Rise\\_and\\_Fall\\_of\\_the\\_Chaos\\_Report\\_Figures](https://www.researchgate.net/publication/220092178_The_Rise_and_Fall_of_the_Chaos_Report_Figures).
- VISCONTI, M. y ASTUDILLO, 2015. *Fundamentos de Ingeniería de Software: Patrones de Diseño*. *Universidad Técnica Federico Santa María*,
- VISUAL PARADIGM, 2018. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. [en línea]. [Consulta: 26 noviembre 2018]. Disponible en: <https://www.visual-paradigm.com/>.
- VISURE, 2018. Reutilización de Requisitos. *Visure Solutions* [en línea]. [Consulta: 24 noviembre 2018]. Disponible en: <https://visuresolutions.es/reusabilidad-de-requisitos/>.
- YOUNG, R.J., 2004. *White mythologies*. S.I.: Routledge. ISBN 1-134-38455-6.
- ZAPATA, C.M., GIRALDO, G.L. y MESA, J.E., 2010. Una propuesta de metaontología para la educación de requisitos. *Ingeniare. Revista chilena de ingeniería*, vol. 18, no. 1, pp. 26-37.

## Anexos

### Anexo 1: Gestionar proyecto

<b>Objetivo</b>	Gestionar los proyectos, teniendo la opción de crearlos, modificarlos y eliminarlos en cualquier momento.
<b>Actores</b>	Analista(Inicia): crea, modifica y elimina los proyectos en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar proyecto”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar un proyecto.

### Anexo 2: Gestionar métricas

<b>Objetivo</b>	Gestionar las métricas, teniendo la opción de crearlas, modificarlas y eliminarlas en cualquier momento.
<b>Actores</b>	Analista(Inicia): crea, modifica y elimina las métricas en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar métrica”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar una métrica.

### Anexo 3: Gestionar métricas

<b>Objetivo</b>	Gestionar las características y subcaracterísticas, teniendo la opción de crearlas, modificarlas y eliminarlas en cualquier momento.
<b>Actores</b>	Analista(Inicia): crea, modifica y elimina las características y subcaracterísticas en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar característica”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar una características o subcaracterísticas.

### Anexo 4: Gestionar usuario

<b>Objetivo</b>	Gestionar los usuarios, teniendo la opción de crearlos, modificarlos y eliminarlos en cualquier momento.
<b>Actores</b>	Administrador(Inicia): crea, modifica y los usuarios en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar usuario”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar un usuario.

### Anexo 5: Gestionar centros de producción

<b>Objetivo</b>	Gestionar los centros de producción, teniendo la opción de crearlos, modificarlos y eliminarlos en cualquier momento.
<b>Actores</b>	Administrador(Inicia): crea, modifica y los centros de producción en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar centros”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar un centros de producción.

### Anexo 6: Gestionar tipos de proyectos

<b>Objetivo</b>	Gestionar los tipos de proyectos, teniendo la opción de crearlos, modificarlos y eliminarlos en cualquier momento.
<b>Actores</b>	Analista(Inicia): crea, modifica y los tipos de proyectos en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar tipo de proyecto”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar un tipos de proyectos.

### Anexo 7: Gestionar ámbito de proyectos

<b>Objetivo</b>	Gestionar los ámbito de proyectos, teniendo la opción de crearlos, modificarlos y eliminarlos en cualquier momento.
<b>Actores</b>	Analista(Inicia): crea, modifica y los ámbito de proyectos en la herramienta.
<b>Resumen</b>	El caso de uso inicia cuando el analista selecciona en el menú la opción “Gestionar ámbito de proyecto”, la herramienta muestra una interfaz, donde permite las acciones de crear, editar y eliminar un ámbito de proyectos.

### Anexo 8: Descripción de las variables de los casos de usos

No.	Nombre de campo	Clasificación	Valor nulo	Descripción
1	Requisito	Campo de texto	No	Es el nombre del requisito que se desea utilizar, el cual permite, letras, números y caracteres alfanuméricos.
2	Característica	Campo de selección	No	Es la clasificación de los requisito no

				funcionales, la cual es seleccionable
3	Subcaracterística	Campo de selección	No	Es la subclasificación de los requisitos no funcionales, la cual es seleccionable
4	Métrica	Campo de selección	No	Es la forma en que se evaluará el requisito no funcional
5	Valor de la métrica	Campo de texto	No	Es el valor cuantificable de la métrica