Universidad de las Ciencias Informáticas

Facultad 3

Extensiones de Visual Paradigm para la generación de productos de trabajo de apoyo a la Especificación de requisitos de software

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

Autores:

Elizabeth Verdecia Martínez

Rayco Fonseca Méndez

Tutora:

Ing. Yaniris Blanco Zamora

La Habana, junio 2015

"Año 57 de la Revolución"

"El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad"

Victor Hugo

DECLARACIÓN DE AUTORÍA:

Declaramos ser los autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año 2015.

_____ _____

Firma del tesista

Elizabeth Verdecia Martínez

Firma del tesista

Rayco Fonseca Méndez

_____

Firma del tutor

Ing. Yaniris Blanco Zamora

Dedicatoria

Mientras más grande es el sacrificio para alcanzar algo, mayor es la satisfacción al obtener el éxito. Siempre se persigue teniendo en mente a aquellos que confían en ti y que esperan que lo logres. Este sueño, transformado en éxito, quisiéramos dedicarlo a las personas más importantes de nuestras vidas:

Elizabeth

A mi mamá y a mi papá por ser mis guías y ejemplos, por estar siempre a mi lado. Por su amor infinito, sus consejos y apoyo constante. Por haber sabido educarme, formarme y darme lo mejor de sí. Por ser un ejemplo de dedicación, entrega y sobre todo por ser mi mayor orgullo.

Rayco

A mis padres y a mi hermano por haberme apoyado siempre, por haber sido tan exigentes conmigo en mis estudios, por su educación, sus consejos y por haber confiado siempre en mí.

Agradecimientos

Elizabeth

A mi mamá, Adaelsis Martínez Álvarez por ser mi razón de ser, por haber creído en mí en momentos en que yo misma no lo hacía, por todo su amor y los sacrificios que ha hecho para que hoy este aquí.

A mi papá, Luis Verdecia Lago, por siempre estar presente y darme el apoyo incondicional que me dio la fuerza para recorrer todo el camino. Por mantenerse al tanto de cuanto ha acontecido en mi vida y formar parte imprescindible de ella.

A Aly y Rey por aguantarme fin de semana tras fin de semana durante todos estos años en la casa. Nunca terminaré de agradecerles.

A mis abuelos, mis tías y mis primas por estar siempre al tanto de todo lo que acontece en mi vida y esperar lo mejor de mí.

A mi hermana Lily, por estar siempre pendiente y ayudarme en todo lo que necesité después de haber entrado a mi vida.

A mi compañero de tesis, porque a pesar de las discrepancias, logramos hacer el mejor equipo de tesis para alcanzar los logros que hoy estamos celebrando.

A mi tutora Yaniris, por ayudarnos siempre que necesitamos de ella y estar pendiente de cada detalle de este trabajo. Al tribunal y oponente por sus críticas constructivas que hicieron posible la culminación de este trabajo con mayor calidad.

A mi familia de la UCI por todos los momentos que pasamos juntos: Dayana, Ory, Ode, Samper, Yasi, Dito, Abelito, Leo, Abel Alexis, Yule, Roly, Dachel y muy especialmente a Choy, por todo el tiempo que me dedicó cuando más lo necesité y por hoy ser una parte tan especial de mi vida.

En fin, a todos los que de una forma u otra tuvieron que ver en la obtención de este logro tan grande…

MUCHAS GRACIAS!!!

Agradecimientos

Rayco

A mi mamá, Lucía Méndez González por su apoyo, su confianza, por creer siempre en mi y estar siemrpe a mi lado, eres la mejor madre del mundo.

A mi papá, Armando Fonseca Navarro por sus consejos, por todo lo que me ha dado, por estar siempre a mi lado cuando más lo necesito, eres el mejor padre del mundo.

A mi hermano, Rayner Fonseca Méndez por estar pendiente a mí y ayudándome en todo lo que podía, sin ustedes tres nunca hubiese alcanzado este sueño los amo a todos.

A mi compañera de tesis, realmente es la mejor compañera de tesis que podría tener, como mismo ella dice: "Somos un buen team", nos entendíamos a la perfección, es la mejor sin dudas.

A mi tutora por apoyarnos, por estar ahí siempre que la necesitábamos, por sus consejos. Agradecer al tribunal por sus exigencias, con el objetivo de aumentar la calidad de nuestro trabajo.

A mi amiga de toda la vida, Ariagna Oramas Interian.

A Tahidy Breto Rodríguez por haber compartido tanto tiempo conmigo durante mi estancia en la universidad.

A Dania y a Dago por ser mi mejor amigo de la universidad.

A mis amigos de la universidad como: Dannel, Yoanis, Marisleydis, Rielo, Elaidis, Roberto, Luis Angel, Carlos José, Freddy, Rojas, Danny, Dailet, Chandy, Adrian.

A mi familia de la UCI y digo familia porque para mí ellos forman parte de mi familia, que son: Choy, Yasiel, Dito, Abelito, Dachel, Leo, Yuleydis, Roly, Yasmín, Dayana, Samper, Venero.

RESUMEN

La solución desarrollada en la presente investigación está constituida por dos extensiones implementadas a la herramienta Visual Paradigm cuyo objetivo fundamental es disminuir el tiempo de realización de los productos de trabajo generados en la actividad de Especificación de requisitos. Con el objetivo de dar solución al problema planteado se desarrollaron dos extensiones que permiten la generación automática del Diagrama de requisitos a partir de las actividades informatizables de un Diagrama de procesos de negocio y el Mapa de navegación a partir de las interfaces asociadas a los requisitos de un Diagrama de requisitos. Luego de analizado el negocio se identificaron los principales conceptos y se definieron los requisitos de software que cumplirían las necesidades y restricciones dadas por el cliente. El desarrollo de la solución fue guiado por la metodología de desarrollo de software Proceso Unificado Ágil y se utilizaron herramientas identificadas a partir de las principales tendencias en la creación de extensiones al Visual Paradigm. Finalizado el proceso de desarrollo, fue realizada la validación de las extensiones mediante la aplicación de técnicas de validación de requisitos, métricas para la validación del diseño y pruebas funcionales, garantizando un producto que satisface al cliente teniendo en cuenta sus necesidades y restricciones.

Palabras claves: Diagrama de requisitos, especificación, extensiones, Mapa de navegación, producto de trabajo, tiempo

Índice

Figuras

Tablas

INTRODUCCIÓN

El uso de las tecnologías de la información constituye un componente fundamental en toda empresa o negocio que desee un crecimiento sostenido aparejado al desarrollo tecnológico. Como resultado de su empleo es posible la obtención y procesamiento de la información oportuna y necesaria para la toma de decisiones, a través de la utilización de sistemas capaces de manejar los procesos de negocio por los cuales se rigen estas instituciones. Actualmente la gestión de procesos de negocio ha recibido gran atención por la comunidad empresarial y por las empresas que desarrollan software para la misma. Estas últimas tienen como objetivo proveer sistemas informáticos robustos y escalables, que permitan la ejecución de los procesos de negocios de determinada organización para lograr así el cumplimiento de sus objetivos.

Los procesos de negocio desempeñan un papel primordial en una organización ya que permiten comprender como opera la misma; además son imprescindibles para el diseño y realización de Sistemas de Información (SI). Un SI se puede definir como un conjunto organizado de personas, procesos y recursos, incluyendo la información y sus tecnologías asociadas, que interactúan de forma dinámica, para satisfacer las necesidades informativas que posibilitan alcanzar los objetivos de una o varias organizaciones [1].

Cuando se va a informatizar un SI es necesario llevar del dominio empresarial al dominio de aplicaciones tomando como base fundamental los procesos de negocio sobre los cuales se soporta la organización. El sistema a desarrollar debe adaptarse al entorno empresarial a informatizar y no al contrario, para que sea útil a los funcionarios de la empresa. Los requisitos de software se deben identificar sobre la base de la gestión de los procesos de negocio y especificar con un grado de claridad que puedan ser comprendidos por los desarrolladores que van a implementar el sistema.

Los procesos de obtención y especificación de requisitos funcionales de software que se han venido realizando hasta el momento para informatizar SI se centran en descubrir los requisitos a partir de los procesos de negocio y luego modelarlos como casos de uso [2, 3]. Siguiendo este modelo no se pueden ver claramente las relaciones que se establecen entre las distintas funcionalidades de un mismo proceso de negocio y no se representan los flujos de trabajo de los procesos que constituyen la organización, lo que puede resultar difícil de entender para el cliente la nueva forma en que le es presentada la información, trayendo como consecuencia que no sepa qué es lo que desea del sistema y no pueda validar si la información es correcta. Por su parte el equipo de desarrollo tiene que cambiar la lógica y estructura de la información que hasta el momento se había definido, por lo que se hace necesario dedicar esfuerzo extra para readaptar la implementación de las funcionalidades con el objetivo de ajustar las relaciones entre ellas.

La Universidad de las Ciencias Informáticas (UCI) es una institución que tiene como objetivo la formación de profesionales, así como el desarrollo de sistemas de software para la informatización de los procesos de la universidad, de instituciones nacionales y extranjeras. Para la elaboración de estos sistemas, inicialmente se empleaba como metodología Proceso Unificado de Desarrollo (RUP), la cual se centra en la arquitectura y es guiada por casos de uso, además está pensada para adaptarse a cualquier proyecto y se tienen en cuenta los procesos de negocio. El uso de esta metodología dificulta la comunicación entre el cliente y el equipo de desarrollo, pues los flujos de trabajo son modelados en un lenguaje incomprendido para el primero. Con el objetivo de solucionar este inconveniente, comienza a ser utilizada la Administración de Procesos de Negocio, cuya finalidad es proveer un lenguaje de fácil entendimiento para las partes implicadas.

En el año 2010 se introduce en el Programa de Mejora (PM), estableciendo un proceso de desarrollo de software basado en RUP. El PM en el área de procesos de Administración de requisitos establece dos variantes: una donde los flujos de modelado del negocio y requisitos se realizan por casos de uso y otra donde el modelado del negocio es guiada por procesos. Al realizar el modelado del negocio guiado por casos de uso sale a relucir el problema de comunicación entre clientes y equipo de desarrollo; mientras que al ser guiado por procesos de negocio solo es posible contar con el documento Especificación de requisitos de software, pues no es elaborado ningún otro producto de trabajo que permita el agrupamiento en estructuras para mostrar las relaciones de prioridad y dependencia existentes entre los requisitos.

Para apoyar el modelado del negocio son utilizadas las herramientas de Ingeniería de Software Asistida por Computadoras (CASE, por sus siglas en inglés), las cuales son herramientas individuales para ayudar al desarrollador o administrador de proyecto durante una o más fases del desarrollo de software (o mantenimiento) [4]. Estas herramientas son utilizadas en el proceso de modelado debido a que permiten entre sus funciones la definición de requisitos de los usuarios, lograr un mejor diseño de los sistemas, mejorar la eficiencia en la programación (por su generación automática de códigos) y brindar además a la administración un mejor soporte en la documentación.

El Visual Paradigm es una suite completa de herramientas CASE que utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés), permitiendo el soporte del ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. La especificación de los requisitos se ve afectada debido a que esta herramienta no cuenta con una funcionalidad que permita a partir de las actividades informatizables de los Diagramas de procesos de negocio, elaborar diagramas para mostrar las relaciones entre los requisitos de software definidos y las interfaces de software correspondientes, así como un mapa para que el usuario navegue por el sistema, afectando el tiempo de desarrollo de los productos de trabajo por parte de

los analistas y la reutilización de las relaciones de dependencia y prioridad definidas en este diagrama.

La ausencia de estas funcionalidades impide a los desarrolladores tener un conocimiento más amplio de las relaciones entre los requisitos a la hora de implementarlos. Además, el proceso de desarrollo de software se vería afectado, al elaborar los analistas estos diagramas paso a paso, empleando en esta tarea un tiempo que podría ser utilizado en otras etapas, ya que en el proceso de realización de los diagramas influyen otros indicadores tales como las características que posea la computadora en la que se trabaje, la complejidad de los diagramas a elaborar y la experiencia del analista.

Las funcionalidades anteriormente mencionadas son tratadas en el Visual Paradigm como extensiones, debido a que no se encuentran entre las funciones definidas por la herramienta. Se entiende por extensión a un programa que puede anexarse a otro para aumentar sus funcionalidades.

Dada la situación problemática antes expuesta se hace necesario ampliar los productos de trabajo para especificar los requisitos de software y con ello disminuir el tiempo de elaboración de los mismos utilizando la herramienta Visual Paradigm, por lo que surge como **problema de investigación**: La herramienta Visual Paradigm no permite generar el producto de trabajo Diagrama de requisitos a partir de las actividades informatizables del Diagrama de procesos de negocio, además no cuenta con un diagrama para mostrar el Mapa de navegación del sistema por proceso, lo que atenta contra el tiempo de desarrollo de estos productos de trabajo.

Se define como **objeto de estudio** la generación automática de productos de trabajo en las herramientas CASE. El **objetivo general** de la investigación es desarrollar dos extensiones para la herramienta Visual Paradigm que permitan generar automáticamente el Diagrama de requisitos a partir de las actividades informatizables del Diagrama de procesos de negocio, además del Mapa de navegación del sistema por proceso que contribuya a disminuir el tiempo de desarrollo de estos productos de trabajo, mientras que el **campo de acción** se enmarca en la generación automática de productos de trabajo en la herramienta CASE Visual Paradigm.

En relación con lo anterior se plantean los siguientes **objetivos específicos**:

- Realizar el marco teórico de la investigación que permita conocer las principales tendencias en cuanto a la generación automática de productos de trabajo en la herramienta CASE Visual Paradigm.
- Realizar el análisis y diseño de dos extensiones a la herramienta Visual Paradigm que permitan generar los productos de trabajo Diagrama de requisitos a partir del Diagrama de procesos de negocio y el Mapa de navegación del sistema.

- Implementar dos extensiones a la herramienta Visual Paradigm que permitan generar los productos de trabajo Diagrama de requisitos a partir del Diagrama de procesos de negocio y el Mapa de navegación del sistema.
- Validar la solución a partir de la ejecución de técnicas de validación de requisitos, métricas para la validación del diseño y pruebas de caja blanca y caja negra, que permitan evaluar los resultados.

Como **idea a defender** se plantea: El desarrollo de dos extensiones para la herramienta Visual Paradigm, que permitan generar automáticamente el Diagrama de requisitos a partir de las actividades informatizables del Diagrama de procesos de negocio, además del Mapa de navegación del sistema por proceso contribuirá a disminuir el tiempo de desarrollo de estos productos de trabajo.

Para dar cumplimiento a las tareas de investigación propuestas se utilizan los métodos científicos:

Métodos teóricos

Analítico-sintético: Para desarrollar la investigación fue necesario descomponer en partes el problema y analizar desde diferentes aristas los conceptos asociados al objeto de estudio, facilitando la comprensión del problema planteado y permitiendo que una vez unidas las partes nuevamente, se sinteticen las ideas y permitan la descripción de las relaciones entre ellas y sus características.

Análisis histórico-lógico: Este método es utilizado para estudiar la evolución histórica y el desarrollo de extensiones a la herramienta Visual Paradigm, así como las tendencias de lenguajes y metodologías empleados en su desarrollo, proporcionando un estudio del funcionamiento de los servicios prestados en las extensiones analizadas y de las tendencias actuales en el proceso de desarrollo de las mismas.

Métodos empíricos

Entrevista*:* Mediante cuestionarios pre-elaborados se realizan preguntas a varios especialistas y analistas con el propósito de obtener la información precisa y necesaria para el desarrollo y la validación de las extensiones que resuelven el problema de la investigación.

Experimento: Este método se utilizó para validar la variable planteada en la idea a defender de la presente investigación, si hubo variación entre sus valores antes y después de comenzar a utilizar las extensiones y si la variación es realmente significativa.

Se define la siguiente estructura del documento, quedando conformado en tres capítulos:

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Se abordan los conceptos esenciales para lograr un mejor entendimiento de las extensiones a implementar. Son analizadas varias extensiones implementadas para la herramienta Visual Paradigm efectuando una comparación entre ellas y destacando las principales tendencias en cuanto a su desarrollo. Se realiza un estudio a la Ingeniería de requisitos (IR) y las actividades que la componen; se describe la metodología de desarrollo, herramientas y tecnologías a utilizar para el desarrollo de las dos extensiones y se estudia el proceso de validación a realizar durante la investigación.

CAPÍTULO 2: ANÁLISIS Y DISEÑO

Se realiza la descripción de las principales definiciones asociadas al dominio del problema. Son identificados los requisitos funcionales y no funcionales. Se muestran los diferentes tipos de diagramas que representan la lógica y funcionamiento del sistema y son evidenciados los patrones de diseño a aplicar en las dos extensiones. Es realizada y evidenciada la validación de los productos de trabajo generados en la disciplina Análisis y diseño.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN

Se abordan los principales elementos para el desarrollo de las etapas de implementación y validación de la solución. Es mostrado el modelo de implementación realizado, destacando el Diagrama de componentes y el de despliegue. Es descrita la estructura y organización de la herramienta Visual Paradigm luego de insertadas las dos extensiones, así como los estándares de codificación empleados en la implementación y son evidenciados los resultados de las distintas pruebas aplicadas a la solución que comprueban la calidad del producto.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan los conceptos esenciales para lograr un mejor entendimiento de las extensiones a implementar. Son analizadas varias extensiones implementadas a la herramienta Visual Paradigm efectuando una comparación entre ellas. Se realiza un estudio de la IR como disciplina de la Ingeniería de software, así como las actividades que la componen, haciendo un mayor énfasis en la etapa de Especificación de requisitos; se describe la metodología de desarrollo, las herramientas y tecnologías a utilizar para el desarrollo de las extensiones y por último se estudia el proceso de validación a realizar durante la investigación.

1.1 Extensiones desarrolladas a la herramienta Visual Paradigm

Una extensión es un programa que puede anexarse a otro para aumentar sus funcionalidades (generalmente sin afectar otras funciones ni afectar la aplicación principal). No se trata de un parche ni de una actualización, es un módulo aparte que se incluye opcionalmente en una aplicación [5]. La herramienta Visual Paradigm además de las múltiples funcionalidades que posee, permite la inserción de extensiones para así aumentar las funciones que realiza y apoyar el proceso de desarrollo de software.

Para comprender el ámbito de la solución a desarrollar se realizó un estudio a un conjunto de extensiones implementadas para la herramienta Visual Paradigm, teniendo como objetivo la evaluación del comportamiento de sus principales características en cuanto a:

- Metodología de desarrollo de software.
- Disciplina en la que se enmarca dentro de la metodología.
- Portabilidad, enfocándose para el análisis en la adaptabilidad del sistema [6, 7].
- Lenguaje de programación.
- Usabilidad, enfocándose para el análisis en la capacidad para ser entendido, la capacidad para ser operado y en la capacidad de atracción [6, 7].

Entre las extensiones estudiadas se encuentran: La extensión para la Administración de Requisitos, la extensión para la generación de las clases de acceso a datos con Doctrine 2.0, la extensión para el soporte al Desarrollo Dirigido por Modelos con Ext JS y la extensión para la evaluación del diseño orientado a objetos.

Extensión 1: Administración de Requisitos [8]

Extensión adicionada a la herramienta CASE Visual Paradigm for UML, de software libre, desarrollada con el objetivo de automatizar el proceso de Administración de requisitos.

Sus principales características son:

- Permite la gestión de proyectos, módulos, requisitos y elementos de la base de datos de cada proyecto.
- Brinda la posibilidad de generar la matriz de trazabilidad.
- Desarrollada utilizando la metodología de desarrollo OpenUp y Java como lenguaje de programación.
- Puede ser ejecutada en diferentes sistemas operativos y presenta una interfaz que es de fácil uso y entendimiento para los usuarios.
- Apoya la disciplina Requisitos, debido a que está enfocada a la actividad de Administración de requisitos.

Extensión 2: Generación de las clases de acceso a datos con Doctrine 2.0 [9]

Extensión desarrollada con el objetivo de facilitar el trabajo en las bases de datos, permitiendo generar desde la herramienta Visual Paradigm las clases de acceso a datos para Doctrine 2.0 a partir de un Diagrama entidad-relación.

- Permite la integración entre los códigos de las entidades modeladas en el Diagrama entidad-relación y el Mapeador de Objetos Relacional (ORM) Doctrine 2.0.
- Fue desarrollada utilizando el lenguaje Java en software libre, pero puede ser utilizado en distintos sistemas operativos tales como Windows, Linux y Nova.
- Durante el ciclo de vida del software es empleado en las etapas de diseño e implementación para el tratamiento de las bases de datos.
- Presenta una interfaz amigable que garantiza la facilidad de uso, entendimiento y atracción a los usuarios.
- Se desarrolló utilizando la metodología de desarrollo de software OpenUP y está enfocada a apoyar la disciplina de Implementación.

Extensión 3: Soporte al Desarrollo Dirigido por Modelos con Ext JS [10]

Extensión añadida a la herramienta Visual Paradigm que permite a partir del Diagrama entidad-relación obtener un Diagrama de clases con sus relaciones asociadas, además permite la generación de código fuente partiendo de la interpretación de los modelos de UML llevados al lenguaje Java Script.

- Desarrollado para ser empleado en software libre a través del marco de trabajo Ext JS, a pesar de ser Visual Paradigm una herramienta multiplataforma.
- Permite el almacenamiento en bases de datos de la información que se maneja para que pueda ser consultada posteriormente.

- Es utilizado durante el ciclo de vida del software en las etapas de análisis, diseño e implementación.
- Fue desarrollada utilizando la metodología de desarrollo OpenUp y el lenguaje de programación Java.
- Presenta una interfaz poco amigable en cuando a capacidad para ser entendido, operado y de atracción.

Extensión 4: Evaluación del diseño orientado a objetos [11]

Extensión añadida a la herramienta Visual Paradigm dirigida a la evaluación de los modelos de diseño orientado a objetos pasando por tres procesos esenciales para su cumplimiento: Recopilación de datos, Cálculo de métricas y Evaluación de métricas.

- Contiene funcionalidades que permiten a partir de un diagrama del diseño orientado a objetos evaluar métricas de calidad, obteniendo como resultado reportes a partir de la información recopilada.
- Es utilizado en la etapa de análisis del ciclo de vida del software.
- Fue desarrollada utilizando la metodología de desarrollo OpenUp y el lenguaje de programación Java.
- Puede ser ejecutada en diferentes sistemas operativos y cuenta con una interfaz atractiva que es de fácil uso y entendimiento para los usuarios.

1.1.1 Valoración de las extensiones estudiadas

A partir del estudio de las extensiones previamente mencionadas fue establecido un análisis comparativo mediante los principales parámetros definidos para el desarrollo de las dos extensiones que constituirán la propuesta de solución. En la Tabla 1 se muestra una comparación entre las características de las extensiones analizadas usando los parámetros establecidos:

Tabla 1. Tabla comparativa de las extensiones estudiadas

Parámetros	Extensiones			
	Extensión 1	**Extensión 2**	**Extensión 3**	**Extensión 4**
Metodología	OpenUp	OpenUp	OpenUp	OpenUp
Disciplina	Requisitos	Implementación	Implementación	Análisis y diseño
Portabilidad	Si	Si	Si	Si
Lenguaje de programación	Java	Java	Java	Java
Usabilidad	Amigable	Amigable	Poco amigable	Amigable

Luego de haber realizado un estudio de las extensiones anteriormente mencionadas se concluye que no cumplen en su totalidad con las necesidades del cliente. La búsqueda realizada permitió llegar a las siguientes conclusiones:

- El 100% de las extensiones analizadas fueron desarrolladas empleando la metodología de desarrollo de software OpenUp, el lenguaje de programación Java y cumplen con la característica en cuanto a portabilidad de ser adaptables, debido a que la herramienta Visual Paradigm puede ser utilizada en distintos sistemas operativos.
- La usabilidad de las extensiones fue evaluada en cuanto a la capacidad para ser entendidas, la capacidad para ser operadas y la capacidad de atracción obteniendo como resultado que el 75% presenta una interfaz amigable, por lo que son entendidas con facilidad por los usuarios finales y ofrecen en todo momento información en forma de mensajes de las acciones realizadas o a realizar. Por otro lado el 25% restante presenta una interfaz poco amigable.
- El 25% de las extensiones se enfocan dentro del ciclo de vida del software en la disciplina de Requisitos, enmarcándose directamente en la actividad de Administración; mientras tanto, el otro 75% es utilizado en las disciplinas de Análisis y diseño e Implementación.

Partiendo del análisis antes expuesto se propone realizar dos extensiones para apoyar dentro de la IR la actividad de Especificación, debido a que ninguna de las extensiones estudiadas está enmarcada en esta disciplina, minimizando así el tiempo de desarrollo de los productos de trabajo que son generados en ella, además, garantizar en su elaboración la obtención de dos extensiones que presenten una interfaz amigable que cumpla con los parámetros de comprensibilidad, operabilidad y atractivo.

Después de realizado el análisis de los sistemas similares, se hace necesario el estudio de la Ingeniería de requisitos de software, para así enmarcar la problemática y definir los elementos necesarios para el desarrollo de la propuesta de solución. En los epígrafes siguientes son caracterizadas tanto la IR como las actividades que la componen.

1.2 Ingeniería de requisitos de software

Un requisito es una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o sus componentes para satisfacer un contrato, estándar, u otro documento impuesto formalmente. Puede definirse también como una cualidad o característica que un determinado sistema o parte de él debe cumplir para satisfacer las necesidades de los clientes o de los usuarios finales. “Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de este” [12].

Los requisitos pueden clasificarse en dos tipos: de usuario y del sistema. Los requisitos de usuario son declaraciones, en lenguaje natural y en diagramas, de los servicios que se espera que el sistema proporcione y de las restricciones bajo las cuales debe funcionar [12]. Los requisitos del sistema pueden clasificarse en funcionales, no funcionales y del dominio. Los requisitos funcionales son los que definen las funcionalidades que el sistema deberá ser capaz de realizar, los no funcionales se refieren a propiedades emergentes que deberá poseer como la fiabilidad, almacenamiento, seguridad, entre otros; por su parte los de dominio reflejan las características y restricciones del dominio de aplicación del sistema.

La IR ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software [3]. A su vez se puede conceptualizar la IR como el proceso de desarrollar una especificación de software. Las especificaciones pretenden comunicar las necesidades del sistema del cliente a los desarrolladores del sistema [13].

La finalidad de la IR es lograr que los requisitos alcancen un estado óptimo antes de llegar a la fase de diseño. Existen disímiles conceptos que se encargan de brindar una definición a la IR, en los cuales se expresa que es un proceso durante el cual se logra compilar, analizar y verificar las necesidades reales del cliente con el sistema y así obtener y entregar una especificación de requisitos del software correcta y completa.

1.3 Actividades de la Ingeniería de requisitos

Roger Pressman en su definición de IR incluye siete actividades: inicio, obtención, elaboración, negociación, especificación, validación y gestión [2], por otro lado, Ian Sommerville establece cuatro subprocesos: estudio de viabilidad, obtención y análisis, especificación y validación [12]; mientras tanto, en la implementación del PM, la UCI adopta una combinación entre ambos criterios: Estudio de viabilidad, Obtención y análisis, Especificación, Validación y Administración de requisitos [14].

Estudio de viabilidad: Establece una comprensión básica del problema, de las personas que quieren la solución, la naturaleza de la solución que se desea y para lograr una efectividad preliminar en la comunicación entre el cliente y el equipo de desarrollo. Según lo establecido en el PM es realizada la planificación de la administración de los requisitos.

Obtención y análisis: En esta actividad los ingenieros del software trabajan con los clientes y los usuarios finales del sistema para determinar el dominio de la aplicación, los servicios que debe proporcionar el sistema, el rendimiento requerido del sistema y la restricción del hardware. Según lo planteado en el PM es determinado el proveedor de requisitos, se prepara el entorno de administración de los requisitos y se realiza la modelación del negocio.

Especificación: Es realizada una descripción completa y detallada de las funcionalidades que debe tener el sistema a implementar, definiendo los requisitos funcionales y no funcionales del mismo. Entre los productos de trabajo que se generan se encuentran la especificación de requisitos de software, la evaluación de requisitos y de casos de uso, la especificación de casos de uso, el documento de salidas del sistema y el diccionario de datos.

Validación: En esta actividad es confirmado por el cliente que los requisitos previamente especificados cumplen con las características que debe presentar el sistema, se verifica si existen requisitos no entendidos, para que de ser así volver a la etapa anterior y al concluir actualizar los planes y especificaciones. Esta etapa es sumamente importante, pues los errores en los requisitos pueden conducir a costos elevados si se descubren durante el desarrollo del sistema o cuando ya se encuentra en uso.

Administración: Es el proceso de comprender y controlar los cambios en los requisitos, además de establecer y mantener un consenso entre el cliente y el grupo del proyecto en el cambio de los requisitos del sistema. Esta actividad es realizada durante todo el proceso de Ingeniería de requisitos.

Especificación de requisitos de software

La Especificación de requisitos constituye una de las actividades más complejas de IR, por lo complicado que puede resultar el proceso de traspasar las necesidades y expectativas del usuario del futuro sistema al analista y este a su vez al equipo de desarrollo. Una pobre especificación de los requisitos funcionales de software provoca que el equipo de desarrollo no entienda correctamente los requisitos y realice una incorrecta implementación, trayendo consigo re-implementación de requisitos, largos períodos de pruebas, decepción por parte del usuario y desacreditación del equipo de desarrollo.

Los requisitos del sistema son versiones extendidas de los requisitos del usuario, son utilizados como punto inicial para la realización del diseño del sistema a implementar. Permiten detallar y definir la forma en que el software cumplirá con los requerimientos dados por el usuario. En teoría, “los requisitos del sistema simplemente deben describir el comportamiento externo del sistema y sus restricciones operativas” [15].

Para la redacción de los requisitos del usuario y en ocasiones la especificación de los requisitos del sistema es utilizado el lenguaje natural. Los requisitos del sistema deben definirse de una forma más detallada que los de usuario, por lo que la comprensión de estas especificaciones puede llegar a ser difícil, pues tanto lectores como redactores deberían utilizar las mismas palabras para el mismo concepto, ya que el lenguaje natural es demasiado flexible y se puede decir lo mismo de

formas completamente diferentes. Debido a esto es realizada la redacción de los requisitos del usuario en un lenguaje que los no especialistas pueden entender, mientras que por otro lado es esencial que la redacción de los requisitos del sistema se realice en una notación más especializada. Esta notación incluye un lenguaje natural estructurado, que depende de la definición de prototipos o plantillas estándares para expresar la especificación de requisitos.

El lenguaje natural estructurado es una forma de redactar los requisitos del sistema donde la libertad del redactor de los requisitos está limitada y donde todos los requisitos se redactan de una forma estándar [12]. Esta forma de especificar los requisitos permite mantener la expresividad y comprensión del lenguaje natural, pero a la vez impone cierto grado de uniformidad. Limita la terminología a utilizar y emplea plantillas para la especificación de los requisitos del sistema.

En esta etapa se desarrolla el producto de trabajo Documento de especificación de requisitos de software. Este documento sirve como contrato entre los clientes y el equipo de desarrollo estableciendo lo que debe proveer el nuevo sistema a implementar. El nivel de detalle a incluir en el mismo depende del tipo de sistema que se vaya a desarrollar.

Existen varios estándares para el Documento de especificación de requisitos, definidos por organizaciones como el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y el Departamento de Defensa de los Estados Unidos. El estándar más conocido es el IEEE 830-1998 [12] y define la siguiente estructura:

1. **Introducción**
   1.1 Propósito del documento de requisitos
   1.2 Alcance del producto
   1.3 Definiciones, acrónimos y abreviaturas
   1.4 Referencias
   1.5 Descripción del resto del documento
2. **Descripción general**
   2.1 Perspectiva del producto
   2.2 Funciones del producto
   2.3 Características del usuario
   2.4 Restricciones generales
   2.5 Suposiciones y dependencias
3. **Requisitos específicos**: incluyen los requisitos funcionales, no funcionales y de interfaz.
4. **Apéndices**
5. **Índice**

Solo con este documento no es suficiente para que el equipo de desarrollo pueda identificar las relaciones y dependencias que existen entre los requisitos definidos, lo que podría provocar errores durante la implementación. Por lo planteado anteriormente se propone realizar mejoras a la herramienta Visual Paradigm con el fin de apoyar el desarrollo del proceso de Especificación de requisitos y lograr con ello ampliar la especificación de los requisitos funcionales basados en los procesos de negocio modelados, detallar con mayor profundidad los requisitos y sus relaciones, además de una mayor comprensión por parte del cliente y el equipo de desarrollo.

Para regir el proceso de desarrollo de las dos extensiones será utilizada una metodología, que indicará paso a paso el camino a seguir. Sus características son detalladas en el siguiente epígrafe.

1.4 Metodología de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos informáticos. Indican paso a paso todas las actividades a realizar para lograr el producto informático deseado, además de las personas que deben participar en el desarrollo de las actividades y qué papel deben de tener. Por otra parte, detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla [16].

Al encontrarse la UCI inmersa en el PM, se define como metodología de desarrollo de software para sus proyectos productivos una variación de Proceso Unificado Ágil (AUP, por sus siglas en inglés) en unión con el modelo CMMI-DEV (Capacidad Madurez Modelo Integración-Desarrollo) v 1.3 [17].

1.4.1 Proceso Unificado Ágil

AUP es una versión simplificada de RUP [18]. Esta 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 [17]. En AUP se establecen cuatro fases que transcurren de manera consecutiva: Inicio, Elaboración, Construcción y Transición, pero en la variación establecida en la UCI quedan definidas tres fases: Inicio, Ejecución y Cierre. El desarrollo de la presente investigación se enmarcará en la fase de Ejecución.

En la fase de Ejecución quedan unificadas las tres fases restantes definidas en AUP (Elaboración, Construcción y Transición). Son ejecutadas las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente, además, se capacita a los usuarios finales sobre la utilización del software.

Disciplinas de AUP

AUP propone siete disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno) [19], pero en la variación establecida por la UCI se decide tener ocho disciplinas, pero a un nivel más atómico que el definido en AUP, las que se desarrollan en la Fase de Ejecución, de ahí que en la misma se realicen iteraciones y se obtengan resultados incrementales [17]. Para la presente investigación se desarrollarán las cinco primeras disciplinas.

1. **Modelado de negocio**: Está destinado a comprender los procesos de negocio de la organización.
2. **Requisitos**: Se centra en desarrollar un modelo del sistema que se va a construir. Comprende la administración y gestión de los requisitos funcionales y no funcionales del producto.
3. **Análisis y diseño**: En esta disciplina (si se considera necesario), los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa y una descripción que sea fácil de mantener. Es modelado el sistema, su forma y arquitectura, para que soporte todos los requisitos, incluyendo los no funcionales.
4. **Implementación**: En esta disciplina se construye el sistema a partir de los resultados obtenidos en el Análisis y diseño.
5. **Pruebas internas**: En esta disciplina es verificado el resultado de la implementación. Se deben desarrollar productos de trabajo de prueba como: diseño de casos de prueba, listas de chequeo y de ser posible, componentes de prueba ejecutables para automatizar las pruebas.

Como apoyo al desarrollo de la propuesta de solución serán utilizados lenguajes y herramientas. A continuación se describen las características de cada uno de ellos.

1.5 Herramientas y lenguajes

En el desarrollo de extensiones para aplicaciones es necesario un conjunto de herramientas y lenguajes que apoyen las fases de la metodología seleccionada. En los sub-epígrafes siguientes se detallan las características de las herramientas y lenguajes que serán utilizadas para desarrollar la propuesta de solución.

1.5.1 Lenguajes

1.5.1.1 Lenguaje Unificado de Modelado 2.0

UML es un lenguaje para especificar, visualizar, construir y documentar los productos de trabajo de los sistemas software, así como el modelado del negocio y otros sistemas no software [20]. Es un lenguaje de propósito general para el modelado orientado a objetos y en ocasiones puede considerarse como un lenguaje de modelado visual que posibilita una abstracción del sistema y sus componentes [21].

UML incluye aspectos conceptuales tales como procesos de negocios, funciones del sistema, expresiones de lenguajes de programación, componentes de software reutilizables y esquemas de bases de datos. Permite especificar y visualizar un sistema aunque no admite la descripción de métodos o procesos. Se utiliza para definir un sistema de software, detallar los productos de trabajo, especificar su documentación y construcción [22].

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones [21]:

- **Visualizar**: UML permite expresar de una forma gráfica un sistema de tal forma que otro lo puede entender.
- **Especificar**: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir**: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar**: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para futuras revisiones.

1.5.1.2 Java 8

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; se pone a disposición del programador para que éste pueda comunicarse con los dispositivos de hardware y software existentes [8].

Java es un lenguaje que se basa en la Programación Orientada a Objetos (POO) desarrollado por Sun Microsystems a principios de los años 90. La POO es un paradigma que basa la estructura de los programas en torno a los objetos [23]. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria [24, 25].

La principal característica de Java es la de ser un lenguaje compilado e interpretado [26]. Todo programa desarrollado en Java ha de compilarse y el código que se genera es interpretado por una

máquina virtual, de este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que son dependientes de la plataforma.

Java es el lenguaje de programación propuesto por la Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) de la herramienta Visual Paradigm, por lo que será el lenguaje a emplear en la implementación de las dos extensiones, a través del Kit de Desarrollo de Java (JDK, por sus siglas en inglés) en su versión 1.8, utilizando la herramienta Netbeans.

1.5.2 Herramientas

1.5.2.1 Visual Paradigm 8.0

Las herramientas CASE son herramientas individuales para ayudar al desarrollador de software o administrador de proyecto durante una o más fases del desarrollo de software (o mantenimiento) [4]. Permiten la automatización de las actividades de gestión de proyectos, la gestión de los productos de los trabajos elaborados a través del proceso de desarrollo y ayudan a los ingenieros en el trabajo de análisis, diseño y codificación.

Visual Paradigm es una herramienta CASE para el desarrollo de aplicaciones que utiliza el lenguaje UML, ofreciendo confiabilidad y estabilidad en el proceso de desarrollo [27, 28]. Soporta el ciclo de vida completo del software: análisis y desarrollo orientados a objetos, construcción, prueba y despliegue. Brinda un diseño centrado en casos de uso y enfocado al negocio, lo que genera un software de mayor calidad, además permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y la generación de documentación.

Entre sus características fundamentales se tienen [10]:

- **Multiplataforma**: Soportada en plataforma Java para Sistemas Operativos Windows, Linux, Mac OS.
- **Interoperabilidad**: Intercambia diagramas UML y modelos con otras herramientas. Soporta la Importación y Exportación a formatos XMI, XML y archivos Excel.
- **Modelado de requisitos**: Captura de requisitos mediante Diagramas de requisitos, modelación de casos de uso y análisis textual.
- **Ingeniería de código**: Permite la generación de código e ingeniería inversa para los lenguajes: Java, C, C++, PHP, XML, Python, C#, VB .Net, Flash, ActionScript, Delphi y Perl.
- **Integración con Entornos de Desarrollo**: Apoyo al ciclo de vida completo de desarrollo de software en IDE como: Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, Jbuilder y otros.
- **Modelado de bases de datos**: Generación de bases de datos y conversiones de Diagramas entidad-relación a tablas de bases de datos, además de mapeos de objetos y relaciones.

Diagramas de procesos de negocio

Entre las distintas funcionalidades con que cuenta el Visual Paradigm se encuentra la realización de Diagramas de procesos de negocio, estos son una representación gráfica para el diseño y el modelado de procesos de negocio visualmente [29]. Esta funcionalidad proporciona una notación gráfica para especificar los procesos de negocio en un diagrama [30].

La utilización de estos diagramas trae ventajas para la organización tales como [30]:

- **Efectividad**: Se concreta lo que ha de llevarse a cabo.
- **Eficiencia**: Reutilización de procesos probados como más eficientes.
- **Consistencia**: Detección de tareas no realizables.
- **Productividad**: Reutilización de procesos más productivos.
- **Ahorro**: Asignación de costes (de tiempo, espacio y económicos) e identificación de los procesos más ventajosos.
- **Calidad**: Mejora general de los procesos.

Por otro lado también genera una serie de ventajas para el analista encargado del proceso [30]:

- Agilización del proceso de desarrollo y por tanto de la carga de trabajo.
- Identificación de errores en fases tempranas.
- Mayor nivel de abstracción.
- Trazabilidad del sistema, por identificación de tareas y su asignación a procedimientos manuales o automatizados.

Entre los principales elementos que son necesarios para componer un modelo de procesos de negocio se encuentran las actividades, que se pueden clasificar en varios tipos para así separar su comportamiento. Las actividades se pueden dividir en dos grupos: informatizables y no informatizables.

Las actividades informatizables se van a definir como [30]:

- **Servicio**: Son las tareas que utilizan algún tipo de servicio, por ejemplo: un servicio Web.
- **Envío**: Son las tareas que envían algún mensaje a un participante externo. Cuando la tarea ha sido terminada, el mensaje debe haber sido enviado.
- **Recibir**: Estas tareas esperan a que llegue un mensaje de un participante externo. La tarea está completada una vez que el mensaje ha sido recibido.
- **Usuario**: Una tarea de usuario es una tarea realizada por una persona con ayuda de una aplicación de software.
- **Reglas del negocio**: Estas tareas facilitan entradas a un motor de reglas de negocio y consiguen la salida del motor.

- **Script**: Implican un guión definido por un modelador o implementador en un idioma que un motor de proceso de negocio puede entender.
- **Referencia**: Una tarea de referencia se refiere a otra tarea por su contenido.

Por su parte, dentro de las no informatizables se encuentran las actividades manuales, que son tareas realizadas sin la ayuda de ningún motor de ejecución de procesos de negocio.

1.5.2.2 Netbeans 7.1

El IDE Netbeans es una herramienta para programadores de código abierto, escrito completamente en Java pero puede servir para cualquier otro lenguaje de programación [24]. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web o para dispositivos móviles. Entre sus características están las administraciones de ventanas, almacenamiento, interfaces y configuraciones de usuario.

Es un buen editor de código**,** multilenguaje, con el habitual coloreado y sugerencias de código, acceso a clases pinchando en el código, control de versiones, localización de ubicación de la clase actual, comprobaciones sintácticas y semánticas, plantillas de código, herramientas de refactorización. Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, y estructura la visualización de manera ordenada, lo que ayuda en el trabajo diario [31].

1.6 Validación de la propuesta de solución

La validación tiene como objetivo asegurar que el sistema desarrollado cumpla con las expectativas y necesidades del cliente. Es más que la comprobación de que el sistema cumple con la especificación de requisitos, busca además establecer que el sistema está hecho para un propósito, o sea, que debe ser lo suficientemente bueno para su uso pretendido [12].

CMMI establece como principio que la calidad de un producto o de un sistema es en su mayor parte consecuencia de la calidad de los procesos empleados en su desarrollo y mantenimiento. El aseguramiento de la calidad es un medio planeado y sistemático para asegurarle a la dirección que los estándares, prácticas, procedimientos y métodos de un proceso se aplican correctamente [32]. Para la comprobación de la calidad serán aplicadas técnicas y métricas para la validación de las etapas del proceso de desarrollo.

Para la validación de los requisitos serán utilizadas las técnicas Revisión técnica formal y Prototipos de interfaz, mientras que la validación del diseño será realizada mediante la aplicación de las métricas Tamaño operacional de clases (TOC) y Relaciones entre clases (RC). La validación del sistema será realizada a través de la generación de casos de prueba que consiste en el desarrollo de pruebas a los requisitos para revelar los problemas que se pueden generar. Estas pruebas pueden clasificarse en dos tipos, de caja blanca y de caja negra.

- **Pruebas de caja blanca**: Se basan en un examen cercano al detalle procedimental. Se prueban las rutas lógicas del software y la colaboración entre componentes, al proporcionar casos de prueba que ejerciten conjuntos específicos de condiciones, bucles o ambos.
- **Pruebas de caja negra**: Son aplicadas a la interfaz del software. Examinan algún aspecto funcional de un sistema que tiene poca relación con la estructura lógica interna del software.

Para la validación de la variable dependiente (tiempo) se aplicará un método experimental, en el que se tomará un grupo de personas y se les pedirá que realicen una misma tarea mediante dos vías. Al terminar el experimento se comparan los resultados obtenidos para comprobar si fue solucionado el problema de investigación planteado.

1.7 Conclusiones del capítulo

Luego de finalizar el presente capítulo se arribó a las siguientes conclusiones:

- Se definió el marco teórico de la investigación a partir del estudio de las principales tendencias en el desarrollo de extensiones a la herramienta Visual Paradigm, evidenciando la necesidad de desarrollar dos extensiones que den solución a la problemática planteada.
- A partir de las principales tendencias estudiadas se definió la metodología AUP para guiar el proceso de desarrollo de la solución, utilizando las herramientas Netbeans y Visual Paradigm y los lenguajes Java y UML.

CAPÍTULO 2: ANÁLISIS Y DISEÑO

En el presente capítulo se realiza la descripción de las principales definiciones asociadas al dominio del problema. Son identificados los requisitos funcionales y no funcionales que se deben tener en cuenta para la implementación de las dos extensiones. Se muestran los diferentes tipos de diagramas que representan la lógica y funcionamiento del sistema, con el objetivo de detallar las actividades para un mejor entendimiento de todo el proceso que se lleva a cabo en la Especificación de requisitos y son evidenciados los patrones de diseño a aplicar en las dos extensiones. Es realizada y evidenciada además la validación de la disciplina Análisis y diseño.

2.1 Modelo de dominio

Para ofrecer la ayuda necesaria a los usuarios, equipo de desarrollo e interesados en entender el contexto en que se encuentran ubicadas las dos extensiones y la posterior representación de la solución, es empleado el diagrama Modelo de dominio donde quedan establecidas las principales definiciones a tratar. El Modelo de dominio es el punto de partida para la etapa de diseño, al crear una representación visual del entorno real de las extensiones, además sirve como mapa explicativo para esclarecer los conceptos en el dominio del problema y los atributos pertenecientes a cada uno de los conceptos [33].

El Modelo de dominio es un subconjunto del modelo de negocio y se realiza cuando no están claros los procesos o cuando no se identifican claramente los actores y trabajadores del negocio. Además, el Modelo de dominio captura los tipos más importantes de objetos que existen, o los eventos que suceden en el entorno donde estará el sistema, se identifican conceptos, se definen y se unen o relacionan en un diagrama de clases UML. En la Figura 1 y la Tabla 2 se muestran el Modelo de dominio y los conceptos asociados al mismo respectivamente.

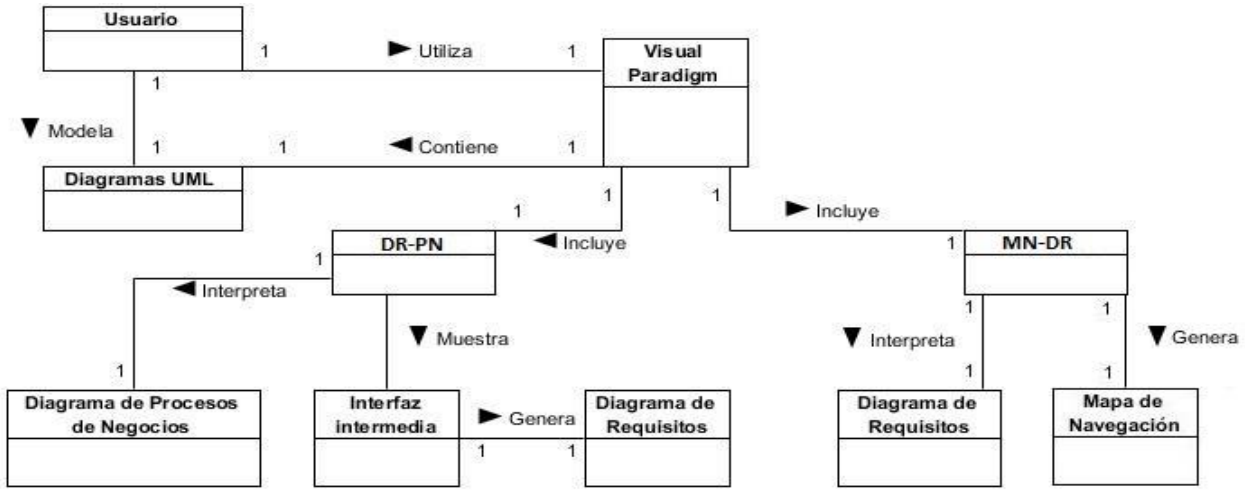


Figura 1. Modelo de dominio

Definición de conceptos del Modelo de dominio

Tabla 2. Conceptos de Modelo de dominio

Elemento	Definición
Usuario	Representa a los usuarios que utilizarán la extensión desde el Visual Paradigm for UML.
Visual Paradigm	Herramienta CASE de modelado para el proceso de desarrollo de software. Entre sus característica incluye la construcción de diagramas tales como el de procesos de negocios e incorporación de funcionalidades a través de extensiones.
Diagramas UML	Representación gráfica en la que se muestran los diagramas que son interpretados por la extensión.
Diagrama de requisitos a partir de los procesos de negocio (DR-PN)	Extensión incluida a la herramienta Visual Paradigm. Contiene las funcionalidades que permiten interpretar la información de un Diagrama de procesos de negocio y generar automáticamente a partir de sus actividades informatizables un Diagrama de requisitos, además de que permite la gestión de los requisitos a insertar en el diagrama.
Mapa de navegación a partir de Diagrama de requisitos (MN-DR)	Extensión incluida a la herramienta Visual Paradigm. A partir de la interpretación de un Diagrama de requisitos permite generar un Mapa de navegación del sistema a través de las interfaces asociadas a los requisitos.
Diagrama de procesos de negocio	Representación gráfica realizada en el Visual Paradigm para el diseño y el modelado de procesos de negocio.
Diagrama de requisitos	Representación gráfica realizada en el Visual Paradigm para presentar la relación entre los requisitos identificados.
Mapa de navegación	Diagrama insertado al Visual Paradigm encargado de mostrar a partir del Diagrama de requisitos las relaciones existentes entre las distintas interfaces asociadas a los requisitos.
Interfaz intermedia	Interfaz insertada a la extensión DR-PN encargada de gestionar los requisitos que van a estar presentes en el diagrama a insertar y las relaciones existentes entre ellos.

La representación del Modelo de dominio constituye el producto de trabajo de entrada para la etapa de Levantamiento de requisitos. En el siguiente epígrafe se ofrece una descripción de las actividades realizadas en esta fase.

2.2 Levantamiento de requisitos

Tomando como base la descripción representada en el Modelo de dominio de las clases más importantes dentro del contexto en que están enmarcadas las dos extensiones, es realizado el levantamiento de los requisitos, proporcionando al equipo de desarrollo una mejor comprensión de las funcionalidades y límites que poseerán las dos extensiones. Como resultado del levantamiento

se obtuvieron los requisitos funcionales y no funcionales que serán implementados en el desarrollo de la solución, detallados en el producto de trabajo Especificación de requisitos del software, que constituye un elemento clave de entrada para la etapa de diseño y la posterior implementación.

Requisitos funcionales

La utilización de técnicas de recopilación de información constituyó un método necesario para realizar el proceso de levantamiento de requisitos, logrando así asegurar y verificar los objetivos propuestos con el desarrollo de las extensiones y satisfacer de esta forma las necesidades del cliente. Entre las técnicas existentes se utilizaron:

- Entrevistas (ver Anexo #1), que fueron aplicadas a especialistas y analistas, posibilitando que cada uno explicara claramente sus necesidades según su experiencia de trabajo en proyectos productivos.
- Tormentas de ideas, para llegar a un consenso entre el cliente y el equipo de desarrollo sobre la mejor manera de obtener la solución y las características que debía poseer para cumplir con las necesidades y restricciones.

En la Tabla 3 se muestra la especificación de los requisitos funcionales (RF) de la propuesta de solución:

Tabla 3. Especificación de los requisitos funcionales

Nº	Nombre	Descripción	Prioridad para el cliente	Complejidad
RF1	Seleccionar Diagrama de procesos de negocio	Garantiza que se seleccione un Diagrama de procesos de negocio.	Alta	Baja
RF2	Extraer relaciones entre las actividades informatizables.	Extrae las relaciones que existen entre las actividades informatizables del diagrama.	Alta	Media
RF3	Crear requisitos a partir de las actividades informatizables y establecer relaciones entre ellos.	Extrae las actividades informatizables del diagrama y las transforma en requisitos manteniendo sus relaciones.	Alta	Alta
RF4	Mostrar relaciones entre los requisitos.	Muestra en la interfaz intermedia las relaciones existentes entre los requisitos identificados del Diagrama de procesos de negocio.	Alta	Media

RF5	Adicionar requisitos.	Permite adicionar nuevos requisitos al Diagrama de requisitos.	Alta	Media
RF6	Eliminar requisitos.	Permite eliminar requisitos del Diagrama de requisitos.	Alta	Baja
RF7	Establecer relación entre requisitos.	Permite establecer nuevas relaciones entre los requisitos del diagrama y modificar las existentes.	Alta	Media
RF8	Eliminar relación entre requisitos	Permite eliminar las relaciones existentes entre los requisitos.	Alta	Media
RF9	Generar diagrama de requisitos.	Muestra el Diagrama de requisitos previamente gestionado.	Alta	Alta
RF10	Seleccionar Diagrama de requisitos	Garantiza que se seleccione un Diagrama de requisitos.	Alta	Baja
RF11	Identificar requisitos con frames[1]	Captura los frames a partir del tránsito de los requisitos.	Alta	Media
RF12	Identificar relaciones entre requisitos	Identifica las relaciones existentes entre los requisitos del Diagrama de requisitos.	Alta	Alta
RF13	Capturar frames a partir de los requisitos y establecer relaciones entre ellos.	Captura los frames asociados a los requisitos y establece las relaciones a partir de las relaciones de los requisitos en el diagrama.	Alta	Media
RF14	Generar Mapa de navegación	Muestra el Mapa de navegación.	Alta	Alta

Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Son fundamentales en el éxito del producto; normalmente están vinculados a los requisitos funcionales, una vez que se conoce lo que el sistema debe hacer se puede determinar cómo ha de comportarse y qué cualidades o propiedades debe tener [34]. Los requisitos no funcionales necesarios para la realización de las extensiones son:

[1] Frames: marco de las ventanas de cada interfaz.

Requisitos de Software

RNF-1: Se deberá tener instalada la herramienta Visual Paradigm for UML 8.0 y el Entorno de Ejecución de Java (JRE, por sus siglas en inglés) 6.0 o superior para que permita la ejecución de programas Java.

Requisitos de Hardware

RNF-2: Se requiere para la ejecución de las dos extensiones un procesador Intel Pentium III a 1.0 GHz o superior. Un mínimo 512 MB de RAM y como mínimo de espacio libre en disco: 1GB (Gigabyte).

Restricciones del diseño e implementación

RNF-3: Se utilizarán para la realización del producto la herramienta de modelado Visual Paradigm for UML en su versión 8.0 y como entorno de desarrollo NetBeans 7.1.
RNF-4: El lenguaje de programación que será empleado para la implementación será Java utilizando el JDK 1.8 siguiendo el paradigma de la POO.

Requisitos de Usabilidad

RNF-5: Facilidad de uso: La interfaz debe ser lo más descriptiva posible, permitiendo que las operaciones a realizar por los usuarios estén bien descritas, de manera que se puedan entender claramente.
RNF-6: La herramienta debe contar con mensajes que permitan mantener informado al usuario de las operaciones realizadas.

Requisitos de Portabilidad

RNF-7: Las extensiones una vez integradas a la herramienta Visual Paradigm for UML podrán ser instaladas y disponer de ellas en diferentes sistemas operativos por ser Visual Paradigm for UML una herramienta multiplataforma.

2.2.1 Descripción de requisitos de software

El documento Descripción de requisitos por procesos es un producto de trabajo generado por la metodología definida y almacenado en el expediente de proyecto de desarrollo de software, donde se describen los requisitos del producto a partir de los elementos del negocio obtenidos. En las Tablas 4 y 5 se muestra un resumen de la descripción textual de los requisitos Adicionar requisito de la extensión DR-PN y Generar Mapa de navegación de la extensión MN-DR, así como sus interfaces de usuario. En el documento Descripción de requisitos por procesos, almacenado en el

expediente de proyecto se muestra la descripción de todos los requisitos funcionales de las dos extensiones a implementar.

Tabla 4. Descripción del requisito Adicionar requisito

Precondiciones	Haber generado la interfaz intermedia	
Flujo de eventos		
Flujo básico		
1	El usuario inserta el nombre del nuevo requisito.	
2	El usuario define si el nuevo requisito se va a comportar como otro existente o no.	
3	El usuario escoge la opción Adicionar.	
4	Concluye el requisito.	
Pos-condiciones		
1	El nuevo requisito es adicionado a la tabla con el resto de los requisitos identificados.	
Flujos alternativos		
Flujo alternativo 3.a El requisito ya existe.		
1	El sistema notifica que el requisito ya existe.	
Flujo alternativo 3.b: El campo nombre está vacío.		
1	El sistema notifica que debe insertar un nombre para el requisito.	
Pos-condiciones		
1	N/A	
Pos-condiciones		
1	N/A	
Validaciones		
1	N/A	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A

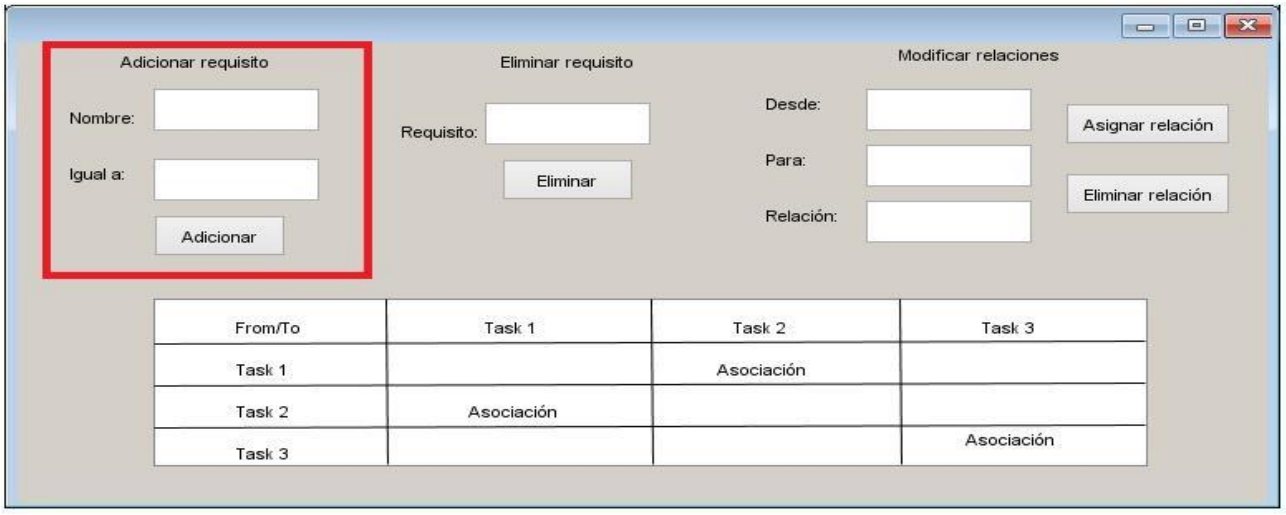


Figura 2. Prototipo de interfaz de usuario del requisito Adicionar requisito

Tabla 5. Descripción del requisito Generar Mapa de navegación

Precondiciones	Han sido capturados los frames y sus relaciones.
Flujo de eventos	
Flujo básico	
1	El sistema permite establecer un nombre al nuevo diagrama.
2	El usuario escoge la opción Generar diagrama.

3	El sistema adiciona un nuevo diagrama con los frames y las relaciones establecidas entre ellos.	
4	Concluye el requisito.	
Pos-condiciones		
1		
Flujos alternativos 2.a: El campo nombre está vacío.		
1	El sistema notifica que debe insertar un nombre.	
Flujos alternativos 2.b: Existe un diagrama con el nombre insertado.		
1	El sistema notifica que el nombre existe y pregunta si desea modificarlo.	
Pos-condiciones		
1	N/A	
Pos-condiciones		
1	N/A	
Validaciones		
1	El sistema comprueba que el campo nombre no esté vacío.	
2	El sistema comprueba que existe un diagrama con el nombre insertado.	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos		
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Figura 3. Prototipo de interfaz de usuario de requisito Generar Mapa de navegación

2.3 Validación de los requisitos

La validación de los requisitos fue realizada con la finalidad de comprobar que los requisitos identificados fueran precisos, consistentes, realistas, verificables, definan lo que el usuario desea del producto final y que los errores que hayan sido detectados fueran corregidos. Para este proceso fueron utilizadas las técnicas Revisión técnica formal y Prototipos de interfaz.

Revisión técnica formal

Una vez terminada la descripción de los requisitos identificados por las necesidades planteadas por el cliente, fueron realizadas una serie de revisiones a cada una de estas descripciones. Ante los errores detectados se procedió a la corrección para comenzar una nueva revisión. Con la utilización

de esta técnica se validó que no existieran errores en el contenido o malas interpretaciones, información incompleta, inconsistencias y que los requisitos no fueran contradictorios, imposibles o inalcanzables, obteniendo como resultado que fueran aprobados aquellos requisitos que estaban descritos de forma correcta, clara y consistente.

Prototipos de interfaz

A partir de las descripciones de requisitos fueron realizadas mediante la herramienta Visual Paradigm prototipos de interfaz de usuario que simulaban la manera en que quedaría el sistema. Con esto se validó que los requisitos estaban en concordancia con las necesidades plasmadas por el cliente. El empleo de esta técnica ofreció como resultado que el cliente tuviera una visión inicial de la estructura de la interfaz de usuario que tendría el sistema.

2.4 Análisis y diseño

La disciplina de Análisis y diseño garantiza la transformación de los productos de trabajo de los requisitos en los productos de trabajo que especifiquen el diseño del software que el proyecto va a desarrollar. Tiene como finalidad: transformar los requisitos en un diseño del sistema en creación, evolucionar una arquitectura sólida para el sistema y adaptar el diseño para que se ajuste al entorno de implementación, con un diseño pensado para el rendimiento [35].

En esta etapa fue modelado el sistema y la forma en que debería quedar al finalizar el proceso de desarrollo para que diera soporte a todos los requisitos, tanto funcionales como no funcionales y así cumplir con las expectativas y necesidades del cliente. Se logró como resultado la obtención de una arquitectura sólida y estable sirviendo como base para la etapa de implementación.

2.4.1 Diseño arquitectónico

Los patrones arquitectónicos ofrecen soluciones a problemas de arquitectura de software, proporcionando una descripción del tipo de relación y restricciones entre los distintos elementos que conforman el diseño. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. Son vistos con un nivel de abstracción mayor que los patrones de diseño, porque muchas arquitecturas diferentes pueden implementar el mismo patrón y por lo tanto compartir las mismas particularidades [9].

La selección de un patrón arquitectónico está en dependencia de las características propias del sistema y pueden variar en dependencia de la conformación del mecanismo de almacenamiento, presentación, controlador o negocio u otros componentes que pueda manejar el sistema [33]. Teniendo en cuenta las características que poseen las dos extensiones se empleará una arquitectura por capas.

Este patrón tiene como objetivo primordial la separación de la lógica del negocio de la lógica de diseño. El estilo en capas es una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. La ventaja principal de la arquitectura en capas es que en caso de ocurrir algún cambio, sólo se modifica el nivel requerido sin tener que revisar entre código mezclado [36].

La arquitectura de las extensiones estaría compuesta por dos capas, la Controladora y la de Presentación. La capa Controladora se encuentra representada principalmente por la clase Controller, en la que estarían implementadas la mayoría de las operaciones para el funcionamiento de las dos extensiones. La capa de Presentación constituida por la clase Interface, es la encargada de interactuar con el usuario, reúne todos los aspectos del software que tiene que ver con las interfaces mediante la utilización de las funcionalidades implementadas en la clase Controller. De esta forma queda evidenciado que la capa de Presentación interactúa con la capa Controladora y desde la filosofía de arquitectura en capas, esto significa que la capa de Controladora presenta un conjunto de funcionalidades para brindar servicios a la capa inmediatamente superior de Presentación. En la Figura 4 se muestra la arquitectura por capas de la extensión.

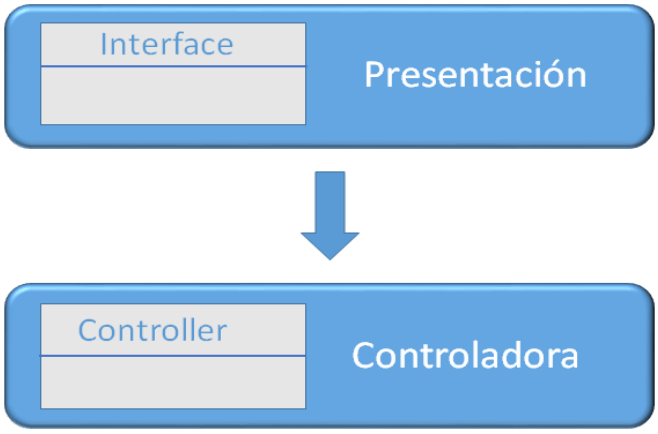


Figura 4. Arquitectura por capas de la extensión

2.4.2 Diseño de clases

El Diagrama de clases es un producto de trabajo fundamental para el análisis y diseño de un sistema. En él, la estructura de clases del sistema se especifica con relaciones entre clases y estructuras de herencia. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama y se modifica para satisfacer los detalles de la implementación [37].

Permite describir gráficamente las especificaciones de las clases de software y las interfaces en una aplicación. Contiene la siguiente información: clases, asociaciones y atributos, interfaces, información sobre los tipos de atributos, navegabilidad y dependencias. En las Figuras 5 y 6 se

muestra el Diagrama de clases del diseño de las extensiones DR-PN y MN-DR de la propuesta de solución:

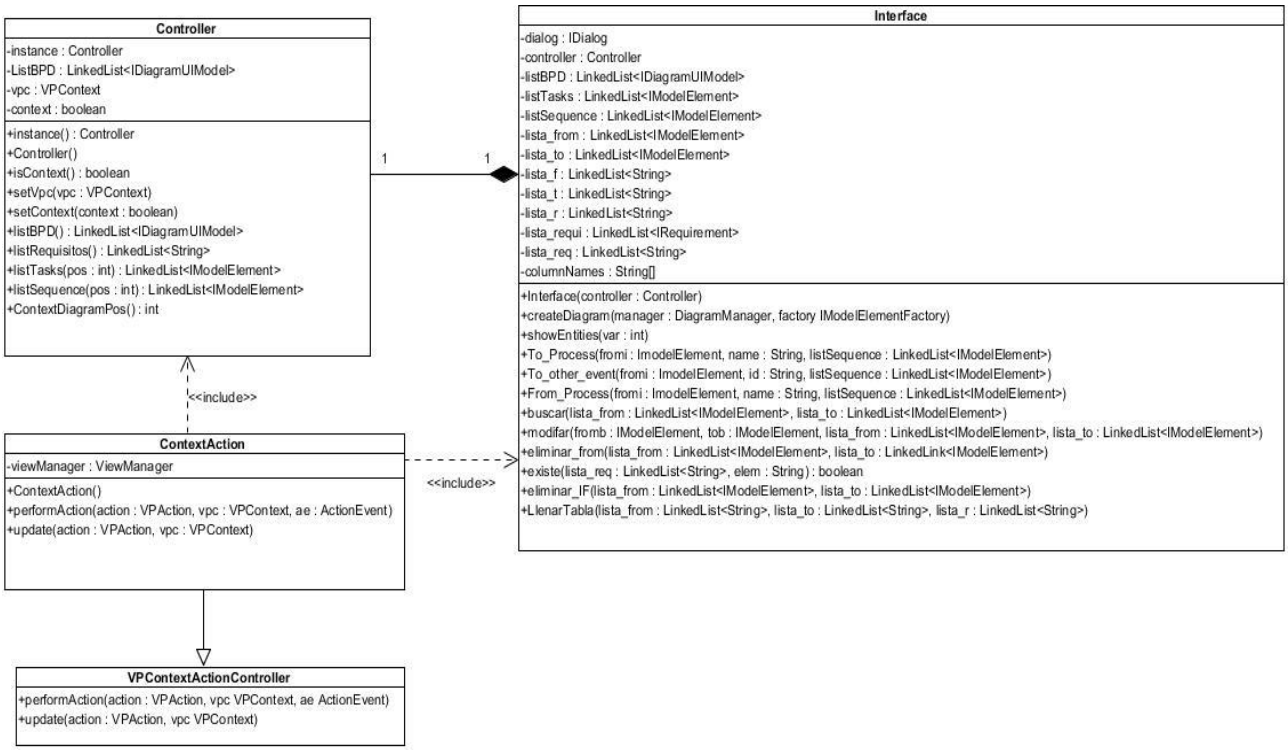


Figura 5. Diagrama de clases del diseño DR-PN

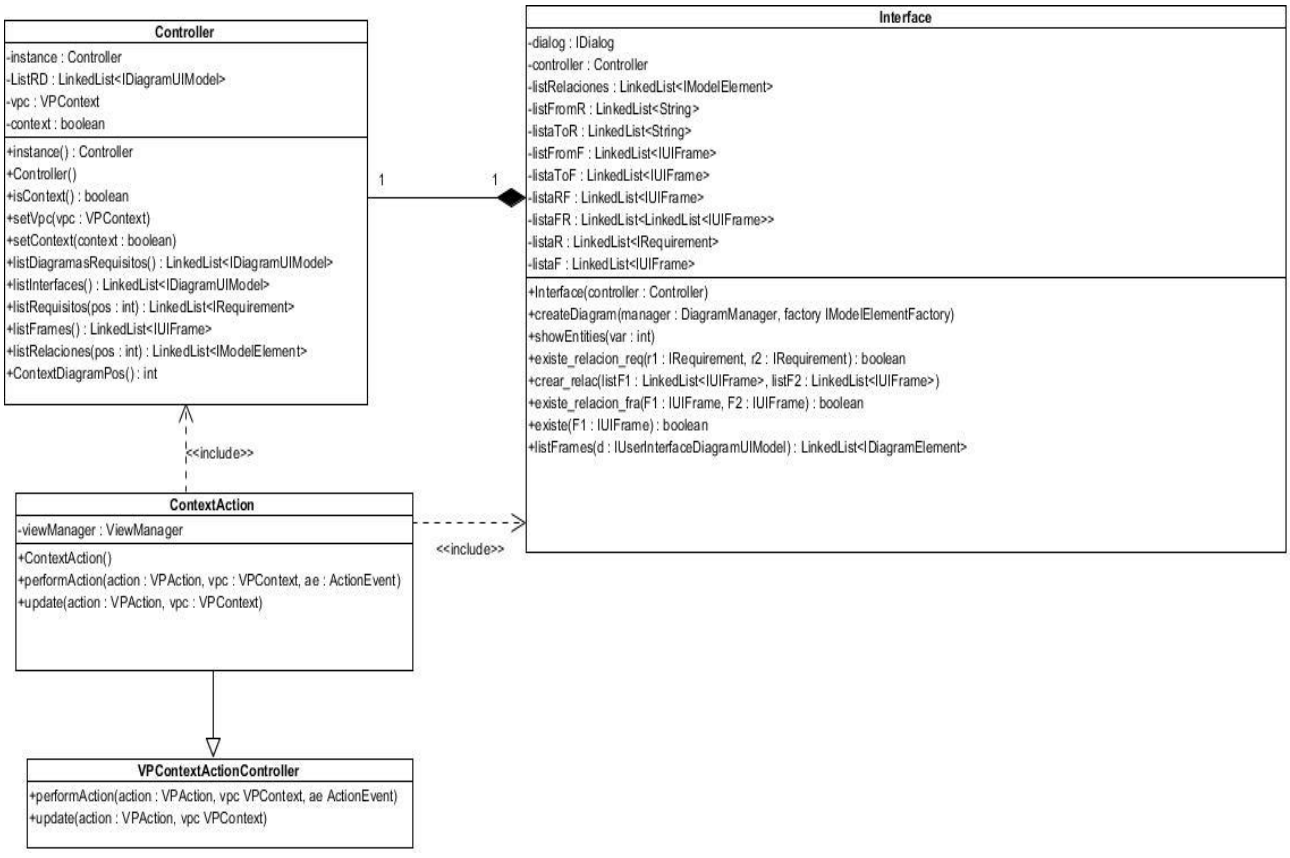


Figura 6. Diagrama de clases del diseño MN-DR

2.4.3 Patrones de diseño

Los patrones de diseño son un mecanismo para describir soluciones generales de problemas de diseño que pueden ser reusadas en la construcción de aplicaciones. Cada patrón prescribe una estructura de clases, sus roles y colaboraciones, además de una adecuada asignación de métodos para resolver un problema de diseño en una manera flexible y adaptable [38].

Existen diversos patrones de diseño utilizados durante el proceso de desarrollo de software que describen los principios fundamentales del diseño de objetos. Son agrupados en dos grandes grupos conocidos como Patrones Generales de Software para la Asignación de Responsabilidades (GRASP, por sus siglas en inglés) y "La Banda de los cuatro" (GOF, por sus siglas en inglés).

Patrones GRASP

Los patrones GRASP son utilizados para describir los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Los patrones empleados en el proceso de desarrollo de la solución son mencionados a continuación:

Experto

La responsabilidad de asignar una labor a la clase que tiene o puede tener los datos necesarios para cumplir determinada responsabilidad, es la solución que pretende dar este patrón ante el problema de cómo realizar la asignación de la forma más eficiente posible.

El patrón experto se ve reflejado en la clase Controller, utilizada por la clase interfaz para asignarle la responsabilidad de las funcionalidades a realizar durante la ejecución de las dos extensiones. Esta clase se identifica como la experta en la información por permitir obtener, almacenar y procesar los datos, en el caso de la primera extensión, del Diagrama de procesos de negocio y así generar el Diagrama de requisitos y en la segunda generar el Mapa de navegación a partir del Diagrama de requisitos.

Controlador

El patrón Controlador es el encargado de asignar la responsabilidad del manejo de uno de los eventos del sistema, a una clase facultada de atender determinada funcionalidad, solucionando así el problema fundamental de este patrón, al saber quién debería ocuparse de dicho evento.

En el Diagrama de clases del diseño de la extensión DR-PN se ve reflejado el patrón Controlador en la clase ContextAction al redefinir las funcionalidades que permiten el control de los eventos proporcionados por la clase VPContextActionController. Esta clase provista por la biblioteca OpenAPI, otorga a la clase ContextAction la capacidad de comunicar el evento generado por el desarrollador al hacer clic derecho en un diagrama del Visual Paradigm en donde se encuentra la opción de "Generar Diagrama de requisitos". Este evento genera como respuesta la ejecución de la

implementación realizada en el método performAction(), que permite crear la interfaz visual de la extensión.

Alta cohesión

La cohesión es la medida en la que un componente se dedica a realizar solo la tarea para la que fue creado, delegando las tareas complementarias a otros componentes. En el diseño realizado, este patrón se manifiesta en la clase Controller, encargada de los procesos de obtención y manejo de los objetos que son obtenidos de los diagramas; y en la clase Interface, que se encarga de la creación y gestión de los nuevos diagramas garantizando así una alta cohesión.

Bajo acoplamiento

El acoplamiento es la medida en que los cambios de un componente tienden a provocar cambios en otro componente. En el diseño realizado se mantiene las mínimas dependencias posibles entre las clases, teniendo en cuenta sus relaciones y prioridades, al ocurrir un cambio en alguna clase de las dos extensiones, no provocaría un impacto relevante pues no se afectarían el resto de las clases garantizando así un bajo acoplamiento.

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. En los Diagramas de clases realizados este patrón se manifiesta en la clase Interface, encargada de crear una instancia de la clase Controller con el objetivo de manejar a través de sí misma las funcionalidades de la clase instanciada.

Patrones GOF

Los patrones GOF constituyen una descripción de clases y objetos que se comunican entre sí, adaptándose para resolver un problema de diseño general en un contexto particular [39]. Los patrones empleados en la implementación son:

Solitario

El patrón Solitario es de tipo creacional. Su principal objetivo es garantizar que una clase tenga una única instancia, proporcionando un punto de acceso global a la misma [40]. Permite realizar refinamientos en las operaciones y en la representación, mediante la especialización por herencia de “Solitario”. Es fácilmente modificable para permitir más de una instancia y para controlar el número de las mismas (incluso si es variable).

Este patrón es utilizado directamente en la implementación de la clase Controller para mantener la constancia entre los objetos obtenidos del Diagrama de procesos de negocio. Esta única instancia de la clase pone de manifiesto el uso del patrón Solitario y su ventaja de evitar la duplicidad a través

del punto de acceso global que es capaz de definir. El siguiente código muestra la aplicación del patrón descrito.

```
public static Controller instance() {
    if (instance == null) {
        instance = new Controller();
    }
    return instance;
}
```

Figura 7. Ejemplo del patrón Singleton en el código

Iterador

El patrón Iterador es de tipo comportamiento. A través de su utilización es posible acceder de forma secuencial a cada uno de los elementos de un objeto agregado sin exponer su representación interna. Permite realizar recorridos sobre objetos compuestos independientemente de su implementación [41].

El Iterador cubre la necesidad de acceder a los elementos de un contenedor de objetos sin tener que trabajar con su estructura interna, facilitando de este modo el acceso a la información [42]. Su utilización tributa al incremento de la flexibilidad porque es posible utilizar nuevas formas de recorrer una estructura con solo modificar el iterador en uso, cambiarlo por otro o definir uno nuevo. También facilita el paralelismo y la concurrencia, pues es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente.

En la extensión propuesta, es aplicado este patrón directamente en la implementación de la clase controladora, en la captura de los Diagramas de procesos de negocios creados, el Iterador permite obtener una lista con los nombres de todos los requisitos existentes en el Visual Paradigm. En el siguiente código se muestra la aplicación del patrón descrito.

```
public LinkedList<String> listRequisitos() {

    LinkedList<String> list = new LinkedList<String>();
    for (int i = 0; i < listDiagramasRequisitos().size(); i++) {
        Iterator elementIterator = listDiagramasRequisitos().get(i).diagramElementIterator();
        while (elementIterator.hasNext()) {
            IDiagramElement element = (IDiagramElement) elementIterator.next();
            if (element.getModelElement() instanceof IRequirement) {
                list.add(element.getModelElement().getName());
            }
        }
    }
    return list;
}
```

Figura 8. Ejemplo del patrón Iterador en el código

2.5 Validación del diseño

Para realizar la validación del diseño fueron aplicadas las métricas TOC y RC que se centran en el recuento de atributos y operaciones para cada clase individual y los valores promedio para el sistema como un todo. Se encargan de medir la calidad de acuerdo a los atributos Responsabilidad, Complejidad de implementación, Reutilización de las clases, Acoplamiento, Complejidad de mantenimiento y Cantidad de pruebas.

Seguidamente son evidenciadas la forma en que fueron aplicadas las métricas y los resultados obtenidos en la extensión DR-PN, en el anexo #3 son presentados los resultados de la aplicación de ambas métricas a la extensión MN-DR.

Tamaño operacional de clases

TOC define los siguientes atributos de calidad [3]:

- **Responsabilidad**: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación**: Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización**: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Los indicadores para la realización de la métrica TOC fueron: clase, cantidad de procedimientos, responsabilidad, complejidad y reutilización. Fue aplicada a un total de 4 clases, 25 procedimientos y un promedio de procedimientos (Prom) de 6.75. Para la definición de las categorías alta, media y baja de los indicadores responsabilidad, complejidad de implementación y reutilización fueron utilizados los siguientes criterios:

Responsabilidad

Alta: cantidad de procedimientos > 2* Prom
Media: Prom < cantidad de procedimientos < 2* Prom
Baja: cantidad de procedimientos < =Prom

Complejidad de implementación

Alta: cantidad de procedimientos > 2* Prom
Media: Prom < cantidad de procedimientos < 2* Prom
Baja: cantidad de procedimientos < =Prom

Reutilización

Baja: cantidad de procedimientos > 2* Prom

Media: Prom < cantidad de procedimientos < 2* Prom

Alta: cantidad de procedimientos < =Prom

Tabla 6. Aplicación de la métrica TOC en la extensión DR-PN

Componente	Clase	Cantidad de procedimientos	Responsabilidad	Complejidad	Reutilización
DR-PN	Interface	12	Media	Media	Media
	Controller	10	Media	Media	Media
	ContextAction	3	Baja	Baja	Alta
	VPContextActionController	2	Baja	Baja	Alta
Total		25			

Al analizar los resultados obtenidos luego de la evaluación, se puede concluir que el diseño propuesto para las dos extensiones posee una calidad aceptable, teniendo en cuenta que el 50% de las clases contiene una menor cantidad de operaciones que la media registrada en las mediciones. Los atributos de calidad evaluados se encuentran en un nivel satisfactorio; de manera que la responsabilidad que posee una clase respecto a otras, la repercusión que tiene un cambio o error en una clase dentro de la solución y el grado de dificultad de implementación del diseño en función del número de operaciones definidas en cada clase fue bajo para un total de dos de las clases, el resto se comporta con una categoría media para todos los indicadores definidos por la métrica viéndose favorecida la reutilización del diseño de software. En la Figura 9 son mostrados los resultados obtenidos tras aplicar la métrica:

Figura 9. Resultados de la métrica TOC en extensión DR-PN

Relaciones entre clases

Las relaciones entre clases están dadas por el número de relaciones de uso de una clase con otras. Los indicadores para la aplicación de la métrica RC fueron: clase, cantidad de relaciones de uso, acoplamiento, complejidad de mantenimiento (CM), reutilización y cantidad de pruebas (CP) [3]. Fue aplicada a un total de 4 clases, 2 relaciones de uso y un promedio de de relaciones de uso (Prom) de 0.5. Para la definición de las categorías alta, media y baja de los indicadores acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas fueron utilizados los siguientes criterios:

Acoplamiento

Ninguno: cantidad de relaciones de uso = 0
Bajo: cantidad de relaciones de uso = 1
Medio: cantidad de relaciones de uso = 2
Alto: cantidad de relaciones de uso >2

Complejidad de mantenimiento

Baja: cantidad de relaciones de uso <= Prom
Media: Prom < cantidad de relaciones de uso < 2* Prom
Alta: cantidad de relaciones de uso > 2* Prom

Reutilización

Baja: cantidad de relaciones de uso > 2* Prom
Media: Prom < cantidad de relaciones de uso < 2* Prom
Alta: cantidad de relaciones de uso < =Prom

Cantidad de pruebas

Baja: cantidad de relaciones de uso <= Prom
Media: Prom < cantidad de relaciones de uso < 2* Prom
Alta: cantidad de relaciones de uso > 2* Prom

Tabla 7. Aplicación de la métrica RC en la extensión MN-DR

Componente	Clase	Cantidad de relaciones de uso	Acoplamiento	CM	Reutilización	CP
DR-PN	Interface	1	Bajo	Media	Media	Media
	Controller	1	Bajo	Media	Media	Media

	ContextAction	0	Ninguno	Baja	Alta	Baja
	VPContextActionController	0	Ninguno	Baja	Alta	Baja
Total		2				

Al analizar los resultados obtenidos luego de la evaluación, se puede concluir que el diseño propuesto para el sistema posee una calidad aceptable, teniendo en cuenta que el 50% de las clases tiene dependencia directa con una sola clase y el resto no posee ninguna dependencia. Los atributos de calidad evaluados se encuentran en un nivel satisfactorio; de manera que el nivel de esfuerzo necesario para sustentar, mejorar o corregir la solución presentada, las dependencias o interconexión de una clase o estructura de clase respecto a otras y el número o grado de esfuerzo necesario para realizar las pruebas de calidad al sistema fue bajo o ninguno para el 50% de las clases, mientras que en el resto los indicadores se comportan entre las categorías baja y media, viéndose favorecida la reutilización del diseño de software. En la Figura 10 se muestran los resultados obtenidos tras la aplicación de la métrica:

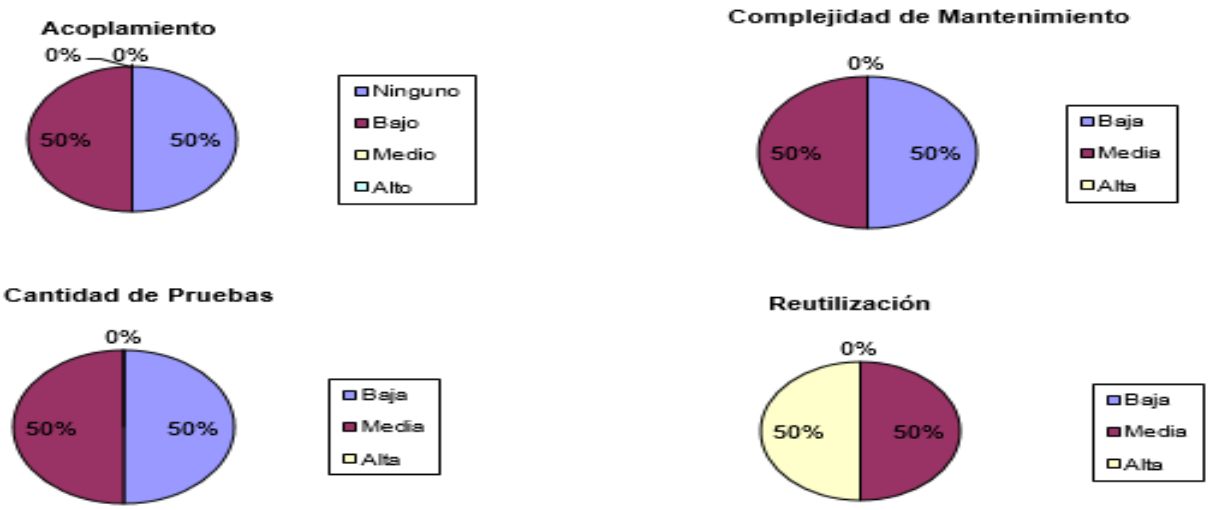


Figura 10. Resultados de la métrica RC en extensión DR-PN

2.6 Conclusiones del capítulo

En este capítulo quedaron expuestos los elementos fundamentales del proceso de Análisis y diseño para la implementación de las dos extensiones, destacando las siguientes conclusiones:

- La representación del modelo de dominio permitió identificar los principales conceptos identificados en el negocio.
- La definición y descripción de los requisitos de software de las dos extensiones permitió establecer las funcionalidades necesarias para que la solución cumpliera con las restricciones y necesidades del cliente, validándolos mediante la aplicación de técnicas.

- La utilización de un patrón arquitectónico y de los patrones de diseño definidos para el desarrollo de la solución facilitó la obtención de un producto con calidad.
- La aplicación de métricas para la validación del diseño permitió evidenciar la calidad del diseño realizado para el desarrollo de las dos extensiones demostrando que los atributos de calidad alcanzaban niveles favorables.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN

En el siguiente capítulo se abordarán los principales elementos para el desarrollo de las etapas de implementación y validación de la solución. Es mostrado el modelo de implementación realizado que pone en práctica el diseño de las dos extensiones, destacando el Diagrama de componentes y el de despliegue. Es descrita la estructura y organización de la herramienta Visual Paradigm luego de insertadas las extensiones, así como los estándares de codificación empleados en la implementación. Son evidenciadas las pruebas aplicadas a la solución con el objetivo de comprobar las funcionalidades en los diferentes escenarios y así verificar en todos los casos que los resultados sean los esperados.

3.1 Implementación

Tomando como punto de partida los productos de trabajo generados en la disciplina de Análisis y diseño es realizada la implementación de la propuesta de solución. Esta etapa tiene como objetivo crear código ejecutable, o sea, código fuente que se puede compilar. Además es creada la documentación necesaria para la instalación, operación y soporte del software y es realizada la integración del sistema, es decir, componer e integrar por separado los componentes de software para que conformen un solo producto entregable [43].

3.1.1 Modelo de implementación

El modelo de implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes podemos encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos [44]. Los elementos que componen el modelo de implementación son el Diagrama de componentes y el Diagrama de despliegue.

3.1.1.1 Diagrama de componentes

Un componente es una parte física de un sistema (módulo, base de datos o programa ejecutable). Es la materialización de una o más clases, ya que una abstracción con atributos y métodos pueden ser implementados en los componentes [13].

Entre los diferentes tipos de componentes utilizados en las extensiones se encuentran:

- **Folder:** Especifica un paquete que contiene en su interior uno o varios componentes.
- **File:** Especifica un componente que representa un documento que contiene código fuente o datos.
- **Library:** Especifica una biblioteca de objetos estática o dinámica.

- **Executable:** Especifica un componente que se puede ejecutar en un nodo.

En las Figuras 11 y 12 se muestran los Diagrama de componentes de las extensiones DR-PN y MN-DR y las Tablas 8 y 9 muestran sus respectivas descripciones:

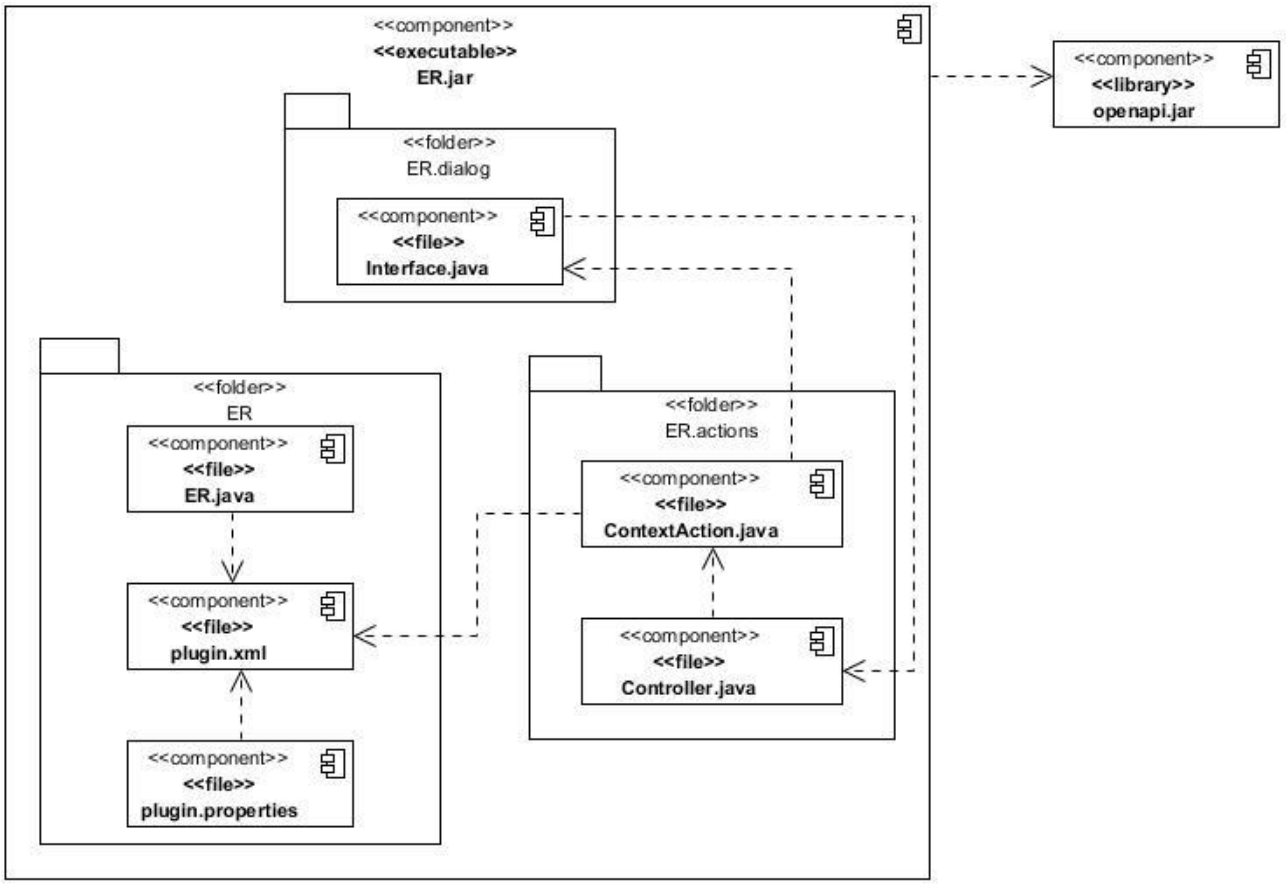


Figura 11. Diagrama de componentes DR-PN

Tabla 8. Descripción de los conceptos del Diagrama de componentes DR-PN

Componente	Descripción
Paquete ER	Paquete que contiene las clases relacionadas con la configuración de la extensión: plugin.xml, ER.java y plugin.properties.
Paquete ER.actions	Paquete que agrupa las clases referentes a las acciones que son accesibles a través del área de diseño del diagrama en Visual Paradigm. Ellas son: ContexAction.java y Controller.java.
Paquete ER.dialog	Paquete que contiene el formulario de la aplicación a través de la clase Interface.java que utiliza la clase Controller.java para la realización de las funcionalidades de la extensión.
ER.java	Carga y descarga el plugin mediante la clase VPPluginInfo que provee el openapi.jar, capturando la información definida en el archivo plugin.xml.
Plugin.xml	Su función consiste en la configuración del plugin, define el nombre de la extensión, descripción, proveedor y las acciones a ejecutar.

Plugin.properties	Define propiedades asociados a los eventos y acciones de la extensión.

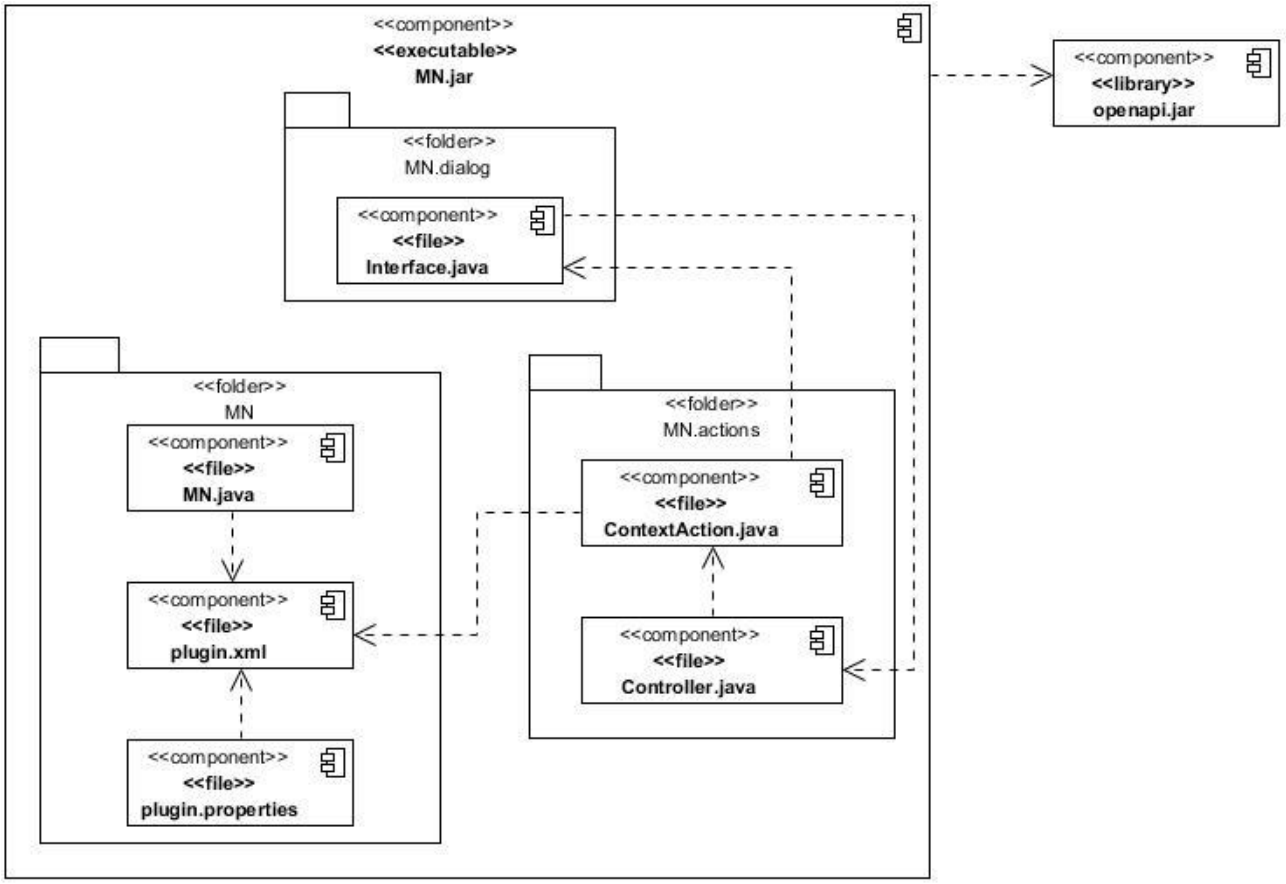


Figura 12. Diagrama de componentes MN-DR

Tabla 9. Descripción de los conceptos del Diagrama de componentes MN-DR

Componente	Descripción
Paquete MN	Paquete que contiene las clases relacionadas con la configuración de la extensión: plugin.xml, MN.java y plugin.properties.
Paquete MN.actions	Paquete que agrupa las clases referentes a las acciones que son accesibles a través del área de diseño del diagrama en Visual Paradigm. Ellas son: ContexAction.java y Controller.java.
Paquete MN.dialog	Paquete que contiene el formulario de la aplicación a través de la clase Interface.java que utiliza la clase Controller.java para la realización de las funcionalidades de la extensión.
MN.java	Carga y descarga el plugin mediante la clase VPPluginInfo que provee el openapi.jar, capturando la información definida en el archivo plugin.xml.
Plugin.xml	Su función consiste en la configuración del plugin, define el nombre de la extensión, descripción, proveedor y las acciones a ejecutar.
Plugin.properties	Define propiedades asociados a los eventos y acciones de la extensión.

3.1.1.2 Diagrama de despliegue

El Diagrama de despliegue muestra las relaciones físicas entre los componentes de hardware y software utilizados en la implementación de sistemas y las relaciones entre sus componentes. Está compuesto por los distintos nodos que componen el sistema y el reparto de los componentes por nodo estará en correspondencia a las características del sistema.

En la representación del Diagrama de despliegue de la extensión se utilizó un único en representación de la PC Cliente. Este nodo se corresponde con el componente de hardware en el que se encuentra instalado la herramienta Visual Paradigm y este a su vez, integra las extensiones DR-PN y MN-DR implementadas. En las Figuras 13 y 14 son mostrados dichos modelos:

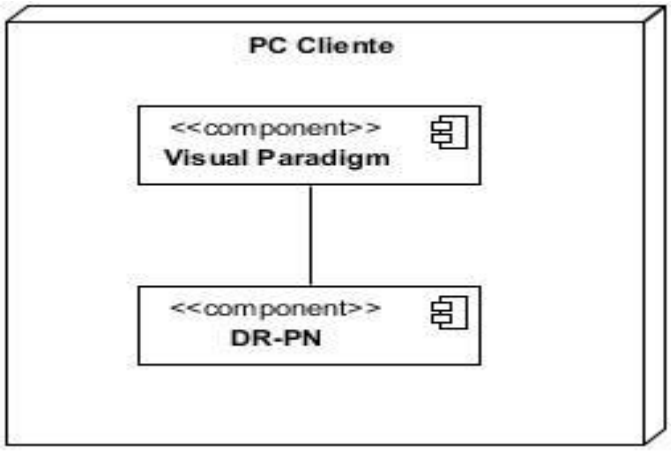


Figura 13. Diagrama de despliegue DR-PN

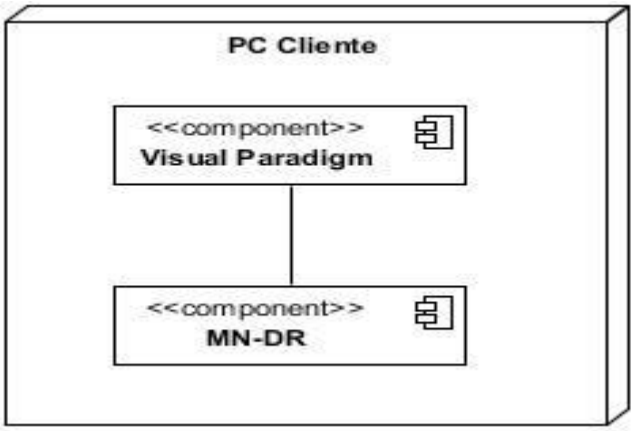


Figura 14. Diagrama de despliegue MN-DR

3.2 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y a la apariencia física del código. Las normas de codificación definidas por la línea de arquitectura están enfocadas a lograr una mejor comprensión y

mantenimiento del código que responda a la complejidad dada por la cantidad de componentes y el alto nivel de integración que puede existir en el desarrollo del software.

La aplicación de forma continua de técnicas y estándares de codificación definidos y la realización frecuente de revisiones al código, junto al empleo de buenas prácticas de programación garantizan una alta calidad en el desarrollo de la solución, además de proporcionar un código legible y reutilizable. Entre los estándares utilizados durante la etapa de implementación se encuentran:

- **Tamaño y organización de las líneas de código**

La longitud de las líneas de código es aproximadamente de 80 caracteres. En caso de que una expresión ocupe más de este rango, podrá romper o dividirse en una nueva línea y deberán estar todas alineadas con respecto a la anterior, para mantener la legibilidad del código.

- **Llaves de apertura y cierre**

Son utilizadas llaves de apertura y cierre, incluso en situaciones en las que son opcionales. Esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos.

- **Declaraciones**

Toda variable local tendrá que ser inicializada en el momento de ser declarada, exceptuando el caso en que su valor inicial dependa de una llamada a otro método en determinado momento de ejecución. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso.

- **Nomenclaturas utilizadas**

PascalCase: establece que los nombres de los identificadores, las variables, métodos y clases están compuestos por una o más palabras juntas, iniciando cada palabra con letra mayúscula y el resto en minúscula. Todas las clases están nombradas siguiendo el estándar PascalCase, de acuerdo al propósito y la función que responden. Ejemplo: las clases Controller y ContextAction.
CamelCase: es similar a PascalCase, la diferencia consiste en que la letra inicial del identificador no comienza con mayúscula. Esta notación fue utilizada para el nombre de funciones, atributos y variables.

- **Notación húngara**

Está basada en definir prefijos para cada tipo de datos según el ámbito de las variables con el objetivo de lograr un mayor entendimiento del nombre de la variable, método o función.

Nomenclatura de las variables: El identificativo de las variables es escrito teniendo en cuenta su objetivo y tomando como referencia el estilo CamelCase, con la primera letra en minúscula. Ejemplo: listTasks que define una lista de tareas, listSequence que define una lista de secuencias.

Nomenclatura de las funciones: El identificativo de los métodos se escribe con la primera palabra en minúscula en dependencia de la función que realizan. Ejemplo: listDiagramasRequisitos(), listInterfaces().

Nomenclatura de los atributos: El identificativo de los atributos se escribe de acuerdo a su objetivo con la primera letra en minúscula. Ejemplo: listBPD.

Luego de establecer los estándares de codificación se procedió a desarrollar la implementación. A continuación se evidencia cómo fue desarrollada esta disciplina para la confección de las dos extensiones.

3.3 Implementación de las extensiones

3.3.1 Inclusión de las extensiones DR-PN y MN-DR a la estructura del Visual Paradigm

La forma en que queda la estructura por carpetas de la herramienta Visual Paradigm luego de incluirle las extensiones DR-PN y MN-DR que conforman la solución es mostrada en la Figura 15. En el presente epígrafe quedará explicado el contenido de las carpetas que componen las extensiones.

Carpeta DR-PN.plugin: Dentro de esta carpeta se encuentran definidas las clases necesarias para el correcto funcionamiento de la extensión DR-PN, incluye dentro de sus clases la Interface y la Controller que permiten capturar los elementos necesarios de un Diagrama de procesos de negocio y generar el Diagrama de requisitos, además de mostrarle al usuario opciones que le permitan gestionar los requisitos a incluir en el diagrama.

Carpeta MN-DR.plugin: Dentro de esta carpeta se encuentran definidas las clases necesarias para el correcto funcionamiento de la extensión MN-DR, incluye dentro de sus clases la Interface y la Controller que permiten capturar las interfaces asociadas a un Diagrama de requisitos y generar un Mapa de navegación para facilitar así la etapa de implementación.

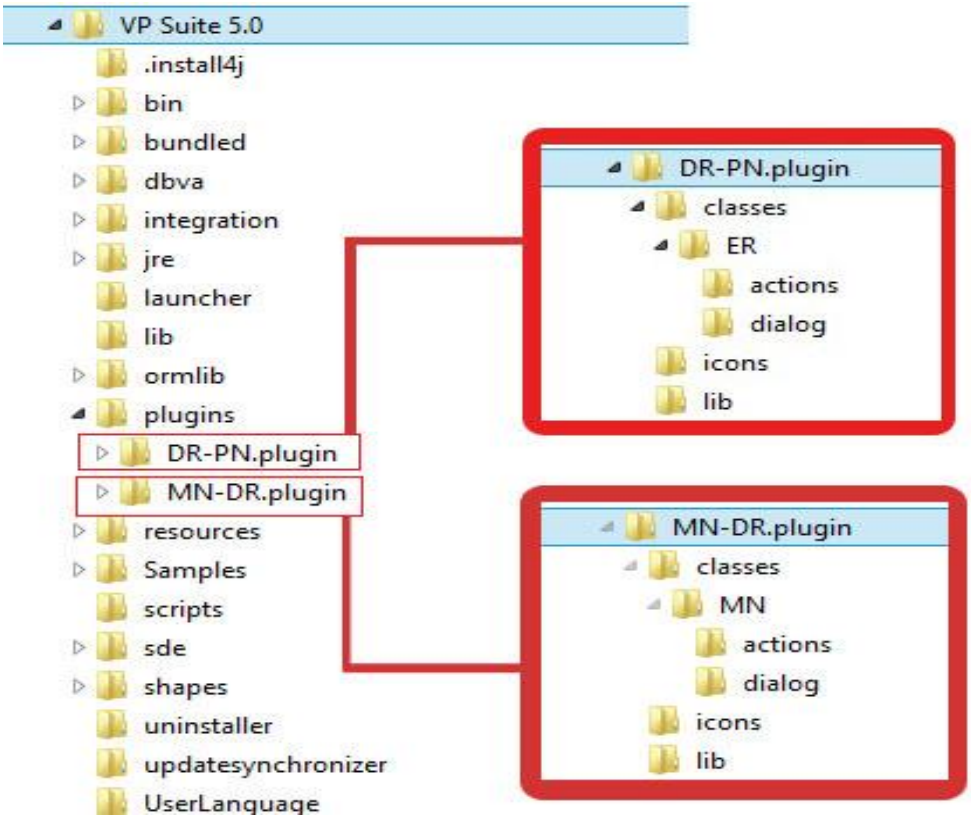


Figura 15. Estructura del Visual Paradigm con las extensiones incluidas

Se desarrolló la misma estructura interna para las carpetas que incluyen ambas extensiones, por lo que solo será evidenciada la explicación de la extensión DR-PN. La carpeta DR-PN.plugin está compuesta por tres carpetas internas: classes, icons y lib.

Carpeta classes: Está contenida por las carpetas **actions** y **dialog**. En la carpeta actions se encuentran los archivos ContextAction.class y Controller.class, que representan las clases controladoras de la extensión. En ellas se encuentran los métodos o funcionalidades que desarrolla la extensión para cumplir con las necesidades del cliente. La carpeta dialog está contenida por el archivo Interface.class que representa la clase interfaz de la extensión.

Carpeta icons: Dentro de esta carpeta están contenidos los íconos que identifican cada una de las extensiones.

Carpeta lib: Dentro de esta carpeta se encuentran las bibliotecas que fue necesario incluir para realizar una correcta implementación de las dos extensiones.

3.3.2 Interfaces de las extensiones

A partir de los productos de trabajo obtenidos en las etapas previas y mediante el uso de los estándares de codificación mencionados, fue realizada la implementación de la solución,

obteniendo las extensiones DR-PN y MN-DR. En su desarrollo fueron obtenidas interfaces gráficas que responden a los requisitos funcionales que fueron definidos antes de implementar. En las Figuras 16 y 17 se muestran las interfaces antes mencionadas:

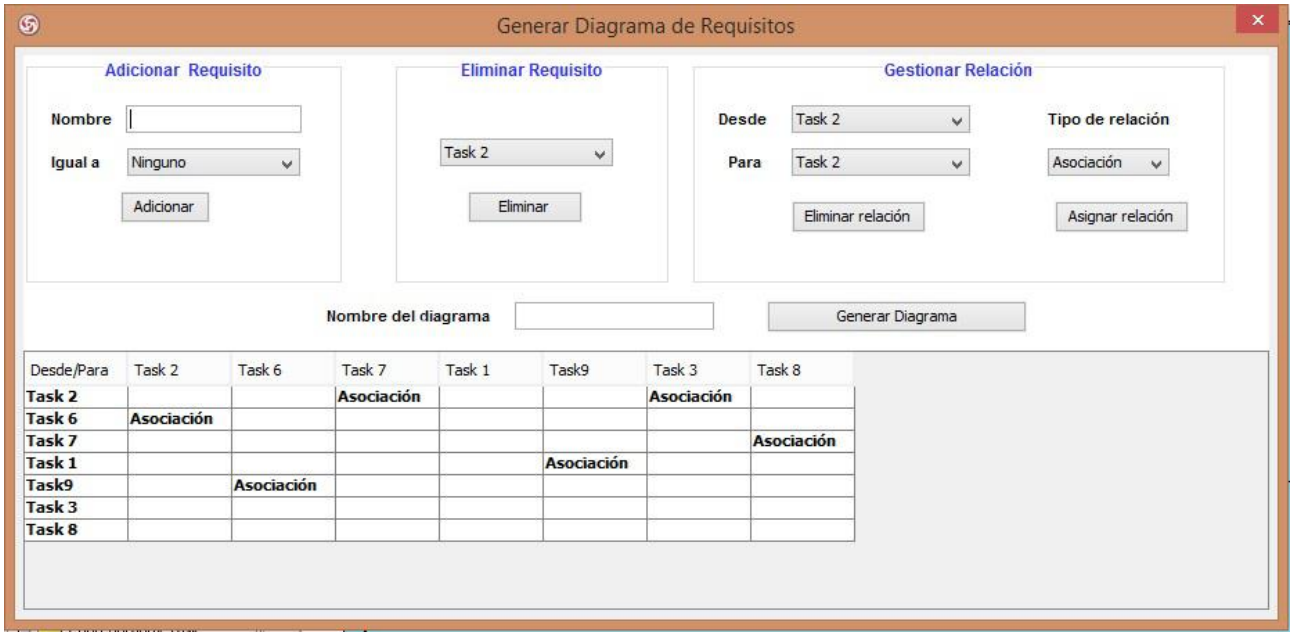


Figura 16. Interfaz gráfica DR-PN

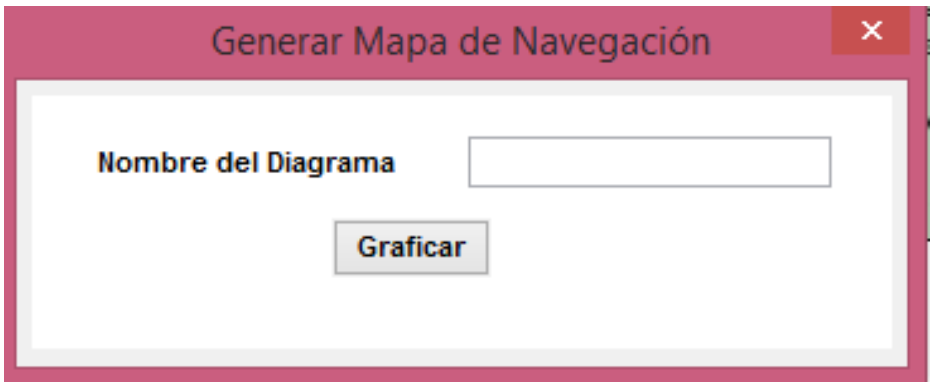


Figura 17. Interfaz gráfica MN-DR

Al finalizar la etapa de implementación se dio comienzo a la fase de validación de las extensiones DR-PN y MN-DR mediante la realización de pruebas de software. En el siguiente epígrafe es descrito el proceso de aplicación de pruebas a la solución.

3.4 Validación de la solución

Durante el proceso de desarrollo de software es necesaria la validación de las distintas fases que lo componen, así como la realización de pruebas con el objetivo de detectar problemas en el sistema, dicho proceso garantiza la calidad del producto y verifica que se cumplan las especificaciones del cliente. En los siguientes sub-epígrafes son evidenciados los procesos de validación y pruebas realizados.

3.4.1 Validación del sistema

Para obtener una evaluación de la calidad interna de las dos extensiones implementadas fueron desarrolladas pruebas de caja blanca y caja negra. Esta calidad será medida en correspondencia con el número de no conformidades que sean detectadas. En el presente sub-epígrafe serán abordados los elementos analizados y los resultados obtenidos en la fase de aplicación de pruebas para medir la variable independiente (extensiones).

3.4.1.1 Pruebas de caja blanca

Las pruebas de caja blanca son un método de diseño que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba [3]. Al utilizar los métodos de prueba de caja blanca se podrán derivar casos de prueba que garanticen que todas las rutas independientes en el módulo se ejecutan por lo menos una vez, se verifiquen los lados verdaderos y falsos de todas las decisiones lógicas y sean ejecutados todos los bucles dentro de sus límites operacionales. Para la comprobación del código de las dos extensiones será utilizada la técnica de la ruta básica o camino básico.

Técnica del camino básico

La prueba del camino básico es una técnica de prueba de caja blanca propuesta inicialmente por Tom McCabe. Permite obtener una medida de la complejidad lógica de un diseño procedimental y utilizar esa medida como guía para la definición de un conjunto básico (diseño de casos de prueba) de caminos de ejecución. Los casos de prueba que son derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa [3].

Para su ejecución es necesario construir el grafo de flujo asociado y calcular su complejidad ciclomática. Una vez identificados los caminos independientes se procede a preparar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Notación del grafo de flujo [45]:

- Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo o grafo del programa.
- Cada círculo denominado nodo del grafo de flujo, representa una o más sentencias procedimentales.
- Un solo nodo puede corresponder a una secuencia de cuadros de proceso y a un rombo de decisión.

- Las flechas del grafo denominadas aristas o enlaces representan flujo de control. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.
- Las áreas delimitadas por aristas y nodos de denominan regiones. Cuando se contabilizan las regiones incluimos el área exterior del grafo, contando como otra región más.

La complejidad ciclomática V(G) puede ser calculada de tres formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

2. Aristas - Nodos + 2, es decir V (G) = A – N + 2.

3. Nodos Predicado + 1 (un nodo predicado es el que representa una condicional if o case, es decir, que de él salen varios caminos).

El método fue aplicado a todo el código de las extensiones DR-PN y MN-DR. En las Figuras 18 y 19 se muestran el código de la funcionalidad showEntities(), encargada de ejecutarse antes de generar el Mapa de navegación del sistema en la extensión MN-DR con las sentencias enumeradas y el grafo de flujo asociado al código.

```
public void showEntities(int var) {
    if (controller.isContext()) { //1
        listR = controller.listRequisitos(var); //2
        if (listR.size() == 0) { //3
            JOptionPane.showMessageDialog(chooser, "No existen requisitos.", "Advertencia", 2); //4
        } else {
            listRelaciones = controller.listRelaciones(var); //5
            if (listRelaciones.size() == 0) { //6
                JOptionPane.showMessageDialog(chooser, "No existe relación entre los requisitos.", "Advertencia", 2); //7
            } else {
                listF = controller.listFrames(); //8
                if (listF.size() == 0) { //9
                    JOptionPane.showMessageDialog(chooser, "No existen frames creados en el VP.", "Advertencia", 2); //10
                } else {
                    existe_transito = true; //11
                }
            }
        }
    }
} //12
```

Figura 18. Funcionalidad showEntities()

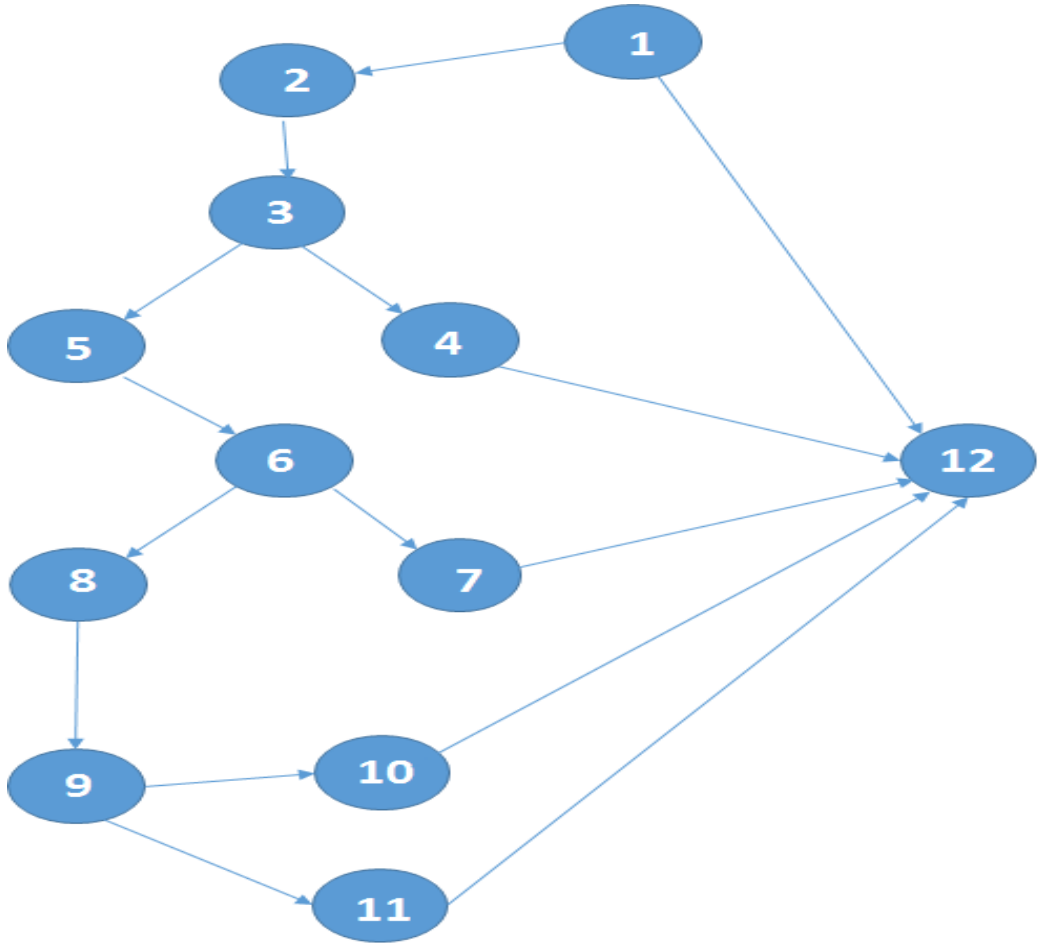


Figura 19. Grafo de flujo asociado al método showEntities()

Cálculo de la complejidad ciclomática:

V (G) = (A - N) + 2 (A: aristas, N: nodos)

V (G) = (15 - 12) + 2 = 5

V (G) = P + 1 (P: nodos predicados)

V (G) = 4 + 1 = 5

V (G) = R (R: regiones)

V (G) = 5

A partir del resultado del cálculo de las fórmulas se obtiene que la complejidad ciclomática de la funcionalidad seleccionada es 5, lo que significa que existen 5 posibles caminos por donde el flujo puede circular, además este valor representa el número de casos de prueba a realizar.

Camino 1: 1,12

Camino 2: 1, 2, 3, 4, 12

Camino 3: 1, 2, 3, 5, 6, 7, 12

Camino 4: 1, 2, 3, 5, 6, 8, 9, 10, 12

Camino 5: 1, 2, 3, 5, 6, 8, 9, 11, 12

Caso de prueba del camino 1:

Descripción: No existe ninguna instancia en la clase Controller de la variable context.

Valores de entrada: La variable var está vacía.

Resultados esperados: No muestra ninguna interfaz.

Caso de prueba del camino 2:

Descripción: Existe una instancia de la variable context, es ejecutado el método listRequisitos() y el sistema muestra una advertencia.

Valores de entrada: La variable var toma valor cero al ejecutar el método listRequisitos().

Resultados esperados: El sistema muestra una advertencia “No existen requisitos”.

Caso de prueba del camino 3:

Descripción: Existe una instancia de la variable context, son ejecutados los métodos listRequisitos() y listRelaciones(), el sistema muestra una advertencia.

Valores de entrada: La variable var toma valor cero al ejecutar el método listRelaciones().

Resultados esperados: El sistema muestra una advertencia “No existe relación entre los requisitos”.

Caso de prueba del camino 4:

Descripción: Existe una instancia de la variable context, son ejecutados los métodos listRequisitos(), listRelaciones() y listFrames, el sistema muestra una advertencia.

Valores de entrada: La variable var toma valor cero al ejecutar el método listFrames().

Resultados esperados: El sistema muestra una advertencia “No existen frames creados en el VP”.

Caso de prueba del camino 5:

Descripción: Existe una instancia de la variable context, son ejecutados los métodos listRequisitos(), listRelaciones() y listFrames, el sistema genera el diagrama.

Valores de entrada: La variable var toma valores distintos de cero al ejecutar los métodos listRequisitos(), listRelaciones() y listFrames().

Resultados esperados: La variable existe_transito toma valor true, permitiendo ejecutar el método para dibujar el mapa.

Al ejecutar los casos de pruebas diseñados para cada camino básico y comparar los resultados obtenidos con los esperados se comprobó que todas las sentencias del código se ejecutan al menos una vez.

3.4.1.2 Pruebas de caja negra

Las pruebas de caja negra son llevadas a cabo sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa, la codificación no se encuentra dentro de sus parámetros a evaluar. Pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se obtiene un resultado correcto. Pretenden encontrar errores en [3]:

- Funciones incorrectas o faltantes
- Errores de interfaz
- Errores en estructuras de datos o en acceso a bases de datos externas
- Errores de comportamiento o desempeño
- Errores de inicialización y término

Para la realización de estas pruebas no fue necesario conocer el funcionamiento interno de las extensiones y su objetivo fundamental fue comprobar que la solución se encontraba acorde a los requisitos definidos. Fueron realizadas una serie de casos de prueba para determinar así que los requisitos estaban parcial o completamente satisfactorios. La evidencia de los diseños de casos de pruebas realizados a las dos extensiones puede ser consultada en los documentos agrupados en la carpeta Prueba almacenada en el expediente de proyecto. En la Figura 20 es mostrado el diseño de casos de prueba realizado a la extensión MN-DR:

Descripción general

Descripción del requisito Generar Mapa de Navegación

Condiciones de ejecución

Cargar vp.vpp en Visual Paradigm/Seleccionar Diagrama de Requerimientos1/Click en el primer requisito/Related Elements /Tránsito a/Manage transit to/Escoger frame de la primera interfaz de usuario/(hacer lo mismo para el segundo requisito)
Click secundario en el area de trabajo del Diagrama de requisitos/Seleccionar opción Generar Mapa de Navegación

SC Insertar nombre al Mapa

Escenario	Descripción	Nombre	Respuesta del sistema	Flujo central
EC 1.1Insertar nombre válido	El sistema debe permitir insertar un nombre al Mapa de Navegación, para luego graficarlo.	V diagrama1 V diagrama1*	El sistema grafica el mapa con el nombre insertado	Se inserta el nombre que tendrá el mapa. Click en el botón Graficar Se grafica el mapa
EC 1.1Crear diagrama con nombre vacío	El sistema no debe permitir crear el mapa con nombre vacío.	I vacío	El sistema notifica que el campo no puede estar vacío: "Debe llenar el campo Nombre del Diagrama"	Click en el botón Graficar El sistema muestra un mensaje de error. No se grafica el mapa

Figura 20. Caso de prueba realizado a extensión MN-DR

Descripción de las variables.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	campo de texto	No	Admite letras, números y caracteres especiales.

Figura 21. Descripción de la variable del caso de prueba

Resultados de las Pruebas

Las dos extensiones fueron probadas en tres iteraciones de pruebas de caja negra. En la primera iteración fueron detectadas once no conformidades de aplicación (diez de la extensión DR-PN y una de la extensión MN-DR), de las que seis fueron de ortografía y cinco de funcionalidad. Las no conformidades fueron resueltas luego de ser identificadas, permitiendo que las dos extensiones pasaran a una segunda iteración de pruebas.

En la segunda iteración se detectaron tres no conformidades de aplicación en la extensión DR-PN, de ellas una fue de funcionalidad y dos de ortografía. Luego de ser detectadas fueron resueltas como en la primera iteración. Al pasar a la tercera iteración no fue encontrada ninguna no conformidad. La evidencia de los resultados antes descritos la constituye el acta de liberación de las extensiones, que puede ser consultada en el anexo #4. En la Figura 22 es mostrada una gráfica con los resultados obtenidos en las tres iteraciones de pruebas:

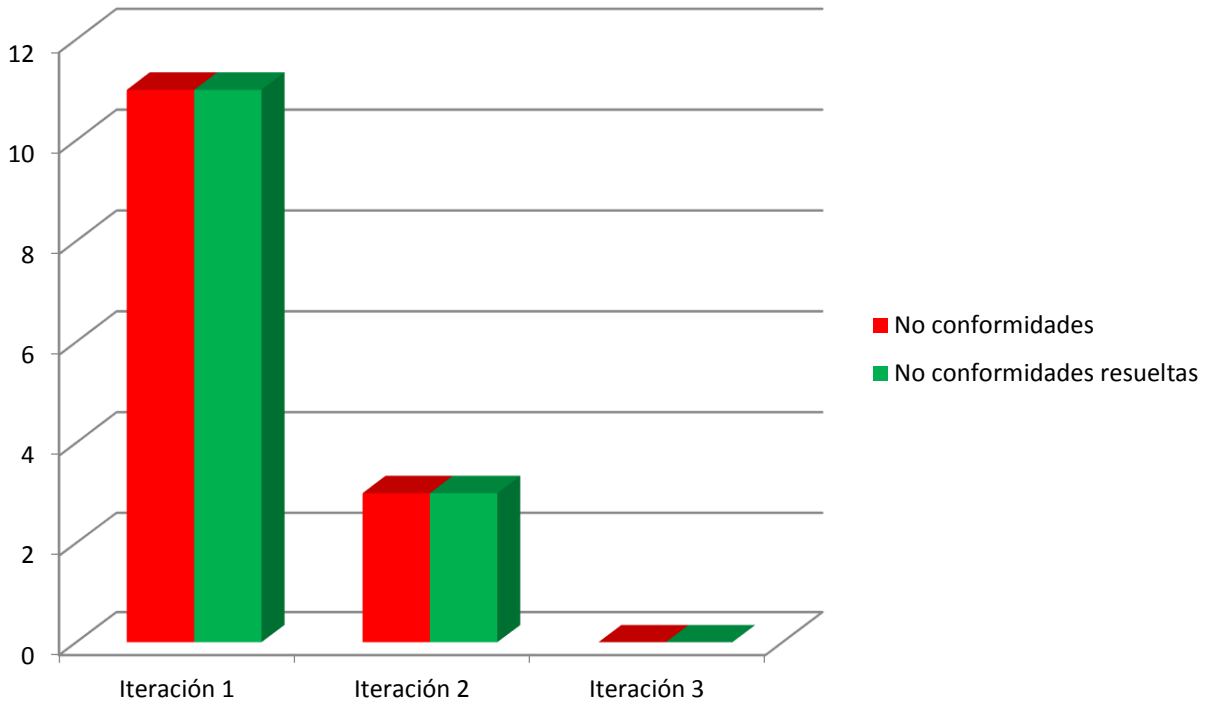


Figura 22. Resultados de las pruebas de caja negra

Además de las pruebas realizadas para probar la calidad de las dos extensiones desarrolladas, se hizo necesario comprobar la variable dependiente de la situación problemática planteada, para así conocer si le fue dado solución al problema. En el siguiente sub-epígrafe es abordada la manera en que se realizó la comprobación.

3.4.2 Validación de la variable dependiente

La presente investigación fue guiada por la siguiente idea: El desarrollo de dos extensiones para la herramienta Visual Paradigm, que permitan generar automáticamente el Diagrama de requisitos a partir de las actividades informatizables del Diagrama de procesos de negocio, además del Mapa de navegación del sistema por proceso contribuirá a disminuir el tiempo de desarrollo de estos productos de trabajo. Para verificar que le fue dado solución al problema definido y el cumplimiento de la idea antes mencionada, se evaluó el comportamiento de la variable tiempo, antes y después de la aplicación de las extensiones DR-PN y MN-DR.

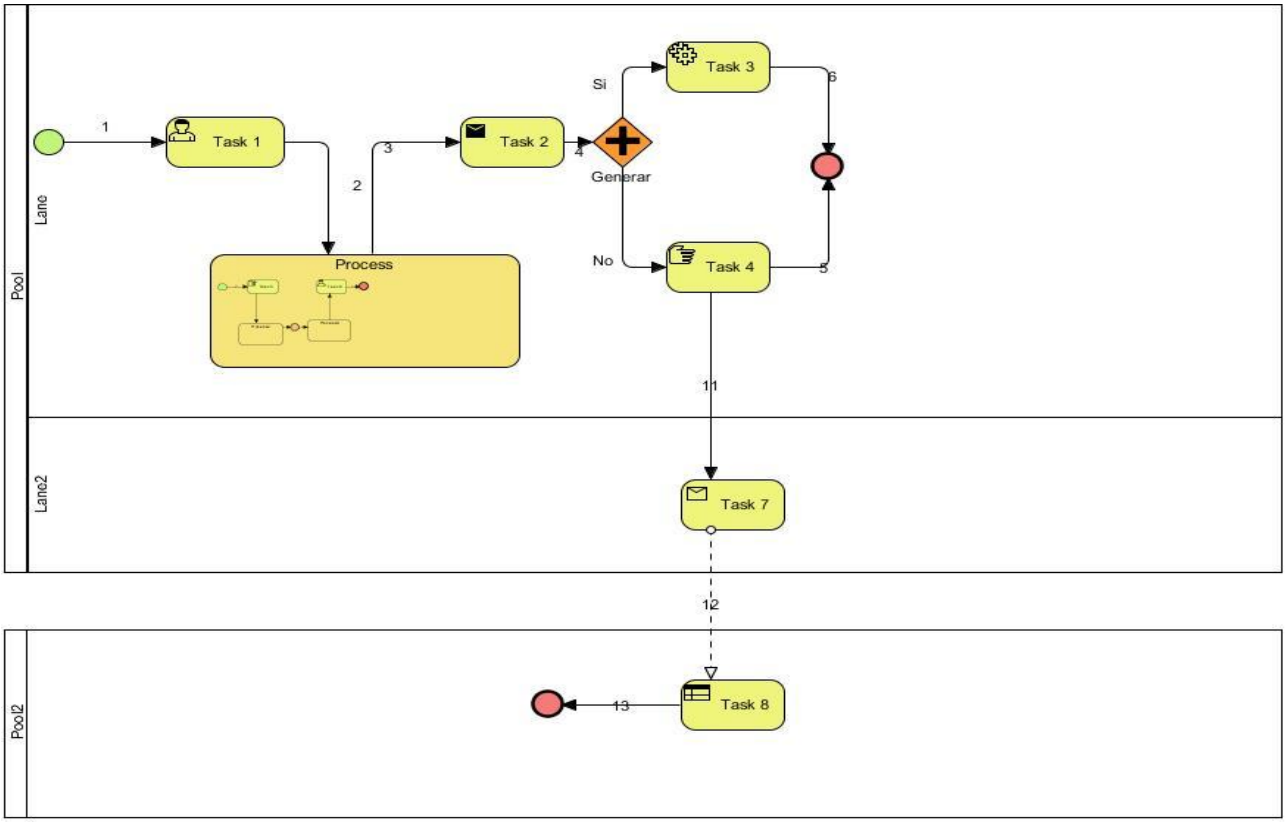
Para realizar la evaluación fue utilizado un método experimental, en el que se escogió un grupo de personas y se les pidió que realizaran una misma tarea de dos maneras diferentes. Para el proceso de selección de las personas fueron definidos los siguientes indicadores:

- Ser estudiante de quinto año o graduado de Ingeniería en Ciencias Informáticas.
- Tener conocimientos para utilizar la herramienta Visual Paradigm.
- Poseer conocimientos básicos de Ingeniería de software.

Finalizado el proceso de selección se obtuvo un grupo de 4 personas en el que dos de ellos eran estudiantes de quinto año y el resto eran graduados de la carrera.

Comprobación de la extensión DR-PN

Para la comprobación del consumo del tiempo en la extensión DR-PN, se seleccionó el grupo de personas y se le pidió que representaran en la herramienta Visual Paradigm el Diagrama de requisitos asociado al Diagrama de procesos de negocio que se muestra a continuación y que luego de realizado, adicionaran dos nuevos requisitos y establecieran entre ellos y el requisito Task 7 la relación Derive:



Luego de terminar esta tarea, se le pidió al mismo grupo de personas que la volvieran a realizar, pero en esta ocasión utilizando la extensión DR-PN. Como resultado se obtuvo una disminución considerable en los valores del tiempo al realizar la tarea utilizando la extensión y haciéndola paso a paso. En la Tabla 10 están evidenciados los resultados antes mencionados, en la que se muestra el tiempo promedio de ejecución en minutos, logrando una mejora de un 90% al utilizar la extensión DR-PN.

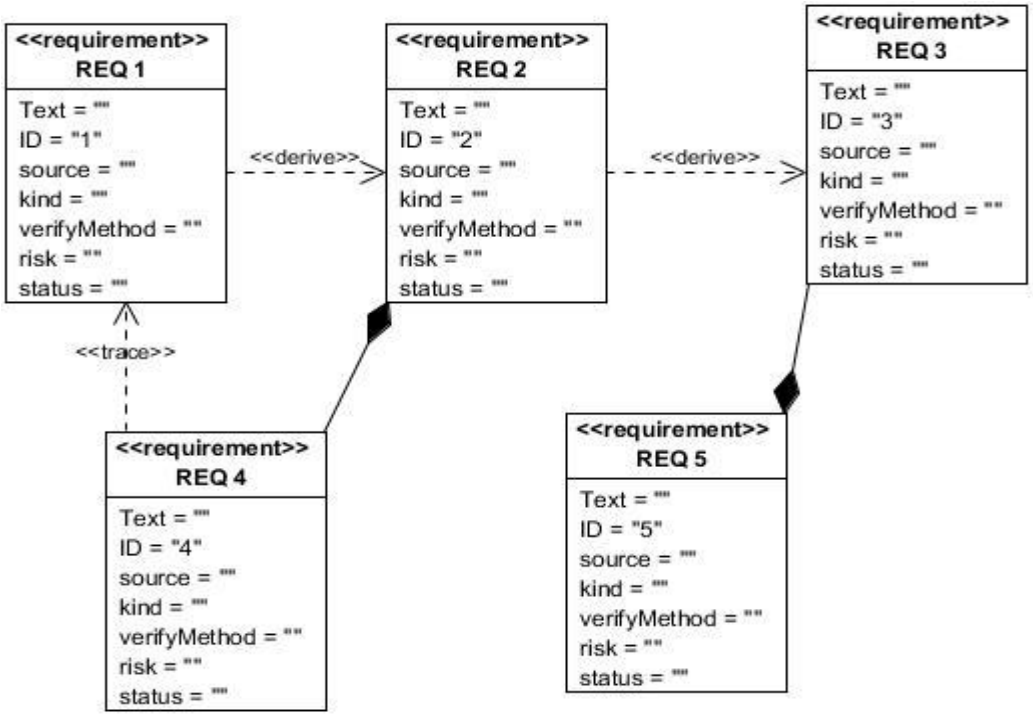
Tabla 10. Variación de la variable tiempo al utilizar la extensión DR-PN

Forma de ejecución / Persona	Paso a paso	Con DR-PN
Persona 1	25	3
Persona 2	22	2
Persona 3	31	3
Persona 4	20	1.5
Promedio de tiempo	24.5	2.4

Comprobación de la extensión MN-DR

Para comprobar que con la extensión MN-DR se logró dar solución al problema de investigación definido se procedió de una manera similar a la anterior. Se seleccionó el grupo de personas y se les mostró un proyecto realizado en la herramienta Visual Paradigm que contenía el Diagrama de

requisitos que se muestra a continuación con las interfaces asociadas a sus requisitos. Luego se les pidió que realizaran un diagrama que mostrara las relaciones existentes entre las distintas interfaces asociadas a los requisitos de este diagrama.



Al finalizar, se les pidió que realizaran la misma acción pero utilizando la extensión MN-DR. Como resultado se obtuvo una disminución en el tiempo de ejecución de la tarea al realizarla mediante la extensión. En la Tabla 11 se muestran los resultados obtenidos en minutos, logrando una mejora de un 96.5% al utilizar la extensión MN-DR.

Tabla 11. Variación de la variable tiempo al utilizar la extensión MN-DR

Forma de ejecución / Persona	Paso a paso	Con MN-DR
Persona 1	22	1
Persona 2	19	0.5
Persona 3	28	1
Persona 4	33	1
Promedio de tiempo	25.5	0,9

Luego de evaluar el comportamiento de la variable tiempo antes y después de la ejecución de ambas extensiones, se puede afirmar que se cumplió el objetivo general y se le dió solución al problema de investigación planteado. La solución desarrollada logra disminuir el tiempo de desarrollo del Diagrama de requisitos a partir de las actividades informatizables del Diagrama de

procesos de negocio y del Mapa de navegación a partir de las interfaces asociadas a los requisitos de un Diagrama de requisitos.

3.5 Conclusiones del capítulo

En este capítulo quedaron presentados los elementos principales de las etapas de implementación y validación de la solución arribando a las siguientes conclusiones:

- La realización del modelo de implementación, que incluye el Diagrama de componentes y el de despliegue facilitó el proceso de implementación de las dos extensiones.
- La definición y utilización de estándares de implementación permitió realizar la codificación de una manera más legible y entendible.
- Mediante la realización de las pruebas de caja blanca y caja negra se comprobó la calidad interna del software, arrojando resultados satisfactorios y demostrando su correcto funcionamiento.
- Mediante la validación de la variable dependiente se comprobó que le fue dado solución al problema de investigación definido.

CONCLUSIONES

Al culminar la presente investigación se puede concluir que fueron desarrolladas todas las tareas necesarias para cumplir el objetivo propuesto, destacando que:

- Se evidenció la necesidad de desarrollar dos extensiones a la herramienta Visual Paradigm que permitieran la generación automática de los productos de trabajo Diagrama de requisitos y Mapa de navegación, utilizando la metodología AUP y las herramientas identificadas a partir de las principales tendencias en el desarrollo de extensiones.
- Se obtuvo el modelo de dominio y los requisitos de software a partir del análisis del negocio, además de los patrones de diseño a emplear en el desarrollo de la solución, validando la definición de los requisitos y el diseño mediante técnicas y métricas.
- La implementación de las extensiones DR-PN y MN-DR fue realizada a partir de los productos de trabajo obtenidos en el Análisis y diseño, permitiendo la generación automática del Diagrama de requisitos y Mapa de navegación.
- Se validaron las dos extensiones a partir de la aplicación de técnicas, métricas y pruebas que garantizaron el correcto funcionamiento de la solución cumpliendo con las necesidades y restricciones dadas por el cliente.

RECOMENDACIONES

Como recomendación del presente trabajo de investigación se tiene:

- Se recomienda perfeccionar las extensiones DR-PN y MN-DR sobre la base de que la interfaz intermedia sea más dinámica en función de que las opciones no se muestren en una sola interfaz, sino que esta se divida por las acciones o requisitos que se quieran acceder, implementándolas desde diferentes interfaces.

Referencias bibliográficas

1. *Gaceta Oficial de la República de Cuba*, M.D. JUSTICIA, Editor. 2011.
2. Labrador Valdés, A.C.M.H., Danaysa ;García Guevara, Julio Jesús *Especificación de requisitos funcionales basada en procesos de negocio*. 2014. La Habana.
3. Pressman, R., *Ingeniería de Software. Un enfoque práctico*. 2005.
4. *Introducción a Herramientas CASE y System Architect*. Universidad Politécnica de Valencia: Valencia.
5. *DICCIONARIO DE INFORMÁTICA Y TECNOLOGÍA*
6. Chua , B.B., Dyson, Laurel Evelyn. *Applying the ISO 9126 model to the evaluation of an e-learning system*. [cited 2015 junio, 6]; Available from: http://www.ascilite.org.au/conferences/perth04/procs/chua.html.
7. Siabato, W. *Métricas aplicadas a los modelos de calidad: caso de uso en los SIG*
8. Pérez Guerra, M., Rondón Milanés,Raúl Alejandro, *Extensión de la herramienta "Visual Paradigm for UML" para la administración de requisitos*. 2012: La Habana.
9. Trujillo Oliva, A., *Extensión de la herramienta Visual Paradigm para la generación de las clases de acceso a datos con Doctrine 2.0*. 2013: La Habana.
10. Ledesma Rodríguez, Y., Boza Roget, Yunier *Extensión de la herramienta "Visual Paradigm for UML" para el soporte al Desarrollo Dirigido por Modelos con Ext JS*. 2011: La Habana.
11. Boizán del Pozo, Y., Galano González, Alexis *Plugin de la herramienta Visual Paradigm para la evaluación del diseño Orientado a Objeto*. 2012.
12. Sommerville, I., *Ingeniería del Software. Séptima edición.* 2005, Madrid: Pearson Addison Wesley.
13. Linares Triana, J.E., *Extensión de la herramienta CASE Visual Paradigm for UML para la Administración de Requisitos.* 2013: La Habana.
14. Ramos Blanco, K., *IPP-3510:2009 Libro de Proceso para la Administración de Requisitos.* 2009.
15. DECSAI, *Especificación de requerimientos. Diseño de bases de datos.* Universidad de Granada.
16. Menéndez, R., Asensio,Barzanallana *Informatica Aplicada a la Gestion Publica*. 2013.
17. Rodríguez, T., *Metodología de desarrollo para la actividad productiva de la UCI.* 2014.
18. Edeki, C. *Agile Unified Process*. 2013.
19. FLORES, E., CORDERO L, JORGE LUIS *METODOLOGIAS AGILES, PROCESO UNIFICADO AGIL (AUP)*. UNIVERSIDAD UNION BOLIVARIANA.
20. Larman, C., *UML y patrones*.
21. Hernández Orallo, E., *El Lenguaje Unificado de Modelado UML*.
22. Rumbaugh, J., Jacobson , Ivar , Booch, Grady *Unified Modeling Language Reference Manual*. 2nd ed. 2004
23. Carballo, Y. *PROGRAMACIÓN ORIENTADA A OBJETOS*.
24. González Valiente, A., Labrador Valdés, Ana Cecilia, et al, *EXTENSIÓN DE VISUAL PARADIGM PARA LA GENERACIÓN DE ARTERFACTOS DE APOYO A LA ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE*. 2014: La Habana.
25. Gosling, J., *The Java Language Specification*. 2nd ed.: Sun Mycrosystem.
26. Belmonte Fernández, O., *Introducción al lenguaje de programación Java. Una guía básica.*
27. *Software.com*, in *Visual Paradigm para UML*.
28. *Tutorial of Visual Paradigm for UML*.
29. Eriksson, H.-E., Penker, Magnus *Business Modeling with UML*.
30. *VP-UML_Users_Guide*.

31. *Gembeta: dev Desarrollo y sotware*. Available from: http://www.genbetadev.com/herramientas/netbeans-1.
32. *Portal de CALISOFT*. Available from: https://calisoft.uci.cu/index.php?option=com_phocadownload&view=category&id=6&Itemid=149.
33. González Hernández, E.P., Torres Ruíz, Allens *Módulo de Reportes del Sistema Informatización Registral de la Cámara de Comercio de la República de Cuba.* 2015: La Habana.
34. *EcuRed* Requisitos no funcionales; Available from: http://www.ecured.cu/index.php/Requisitos_No_Funcionales.
35. *Análisis y diseño*. 2014 [cited 2015 14, marzo]; Available from: http://www.ecured.cu/index.php/Flujo_de_Trabajo_An%C3%A1lisis_y_Dise%C3%B1o.
36. Shaw, M., Garlan, David *An introduction to Software Architecture*. New Jersey: World Scientific Publishing Company.
37. *Modelado de Sistemas con UML*. Available from: ftp://ftp.inf.utfsm.cl/pub/Linux/Docs/LuCaS/Tutoriales/doc-modelado-sistemas-UML/multiple-html/x219.html.
38. Solarte, M.; Available from: http://es.slideshare.net/Enf2IST/introduccin-a-los-patrones-de-diseo.
39. *Reutilización del Software. Patrones de diseño*.
40. *Microsoft*. Available from: https://msdn.microsoft.com/es-es/library/bb972272.aspx.
41. *Escuela Politécnica Superior*.
42. Cabrera, J., Cruz, Azucena de la, Santillan, Daniel *Slideshare*. 2014.
43. *Conceptualización del proceso de implementación de software: perspectivas ágil y disciplinada*. Available from: http://erevistas.saber.ula.ve/index.php/cienciaeingenieria/article/viewFile/1147/1102.
44. Hernandez, L. *MODELO DE IMPLEMENTACIÓN*. Available from: http://ithleovi.blogspot.com/2013/06/unidad-5-modelo-deimplementacion-el.html.
45. *ESTRATEGIAS DE PRUEBA DEL SOFTWARE*. 2011.

Anexos

Anexo #1: Entrevista realizada a especialistas y analistas.

Nombre de la persona: _____

Cargo: _____

Departamento o área: _____

1. ¿Qué herramienta CASE es utilizada para el modelado de los productos de trabajo en el proceso productivo de desarrollo de software?
2. ¿Qué tiempo toma realizar el Diagrama de requisitos a partir del Diagrama de procesos de negocio?
3. ¿Considera usted que a partir de Diagramas de procesos de negocios de alta, media y baja complejidad, la calidad en los Diagramas de requisitos que se generan es la misma?
4. ¿Cuentan en el proyecto con alguna funcionalidad o herramienta en la que se pueda representar en forma de diagrama las relaciones que existan entre las interfaces de usuario asociadas a los requisitos de un Diagrama de requisitos?
5. ¿Sabe usted lo que es un Mapa de navegación?
6. ¿Sería útil la adición de una funcionalidad al Visual Paradigm que permita mostrar un Mapa de navegación del sistema? ¿Ahorraría tiempo en la etapa de implementación?

Anexo #2: Prototipos de los requisitos funcionales

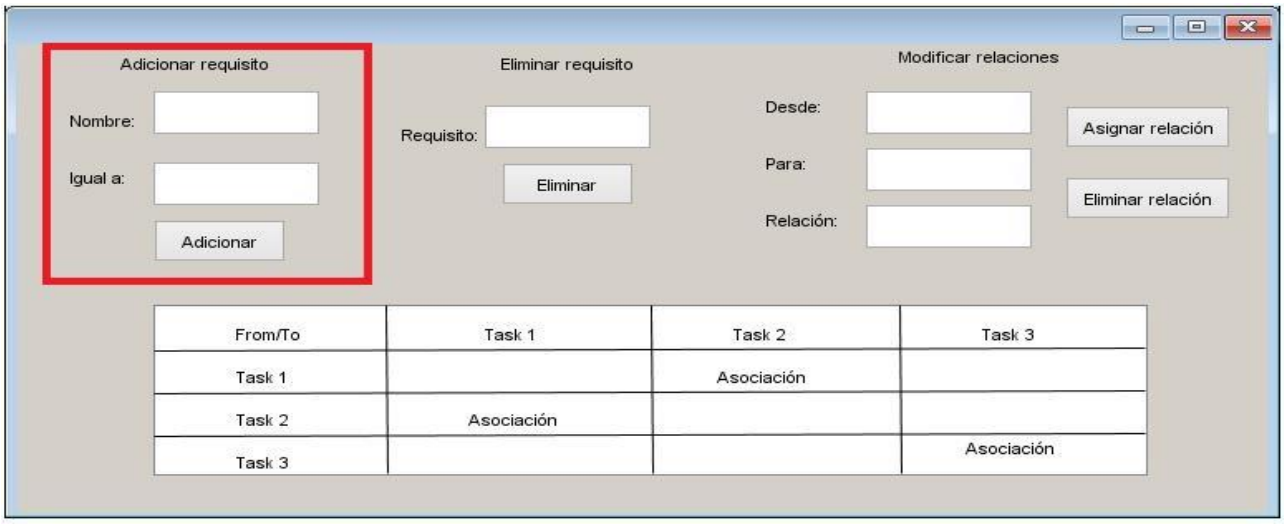


Figura 23. Prototipo del requisito Adicionar requisito

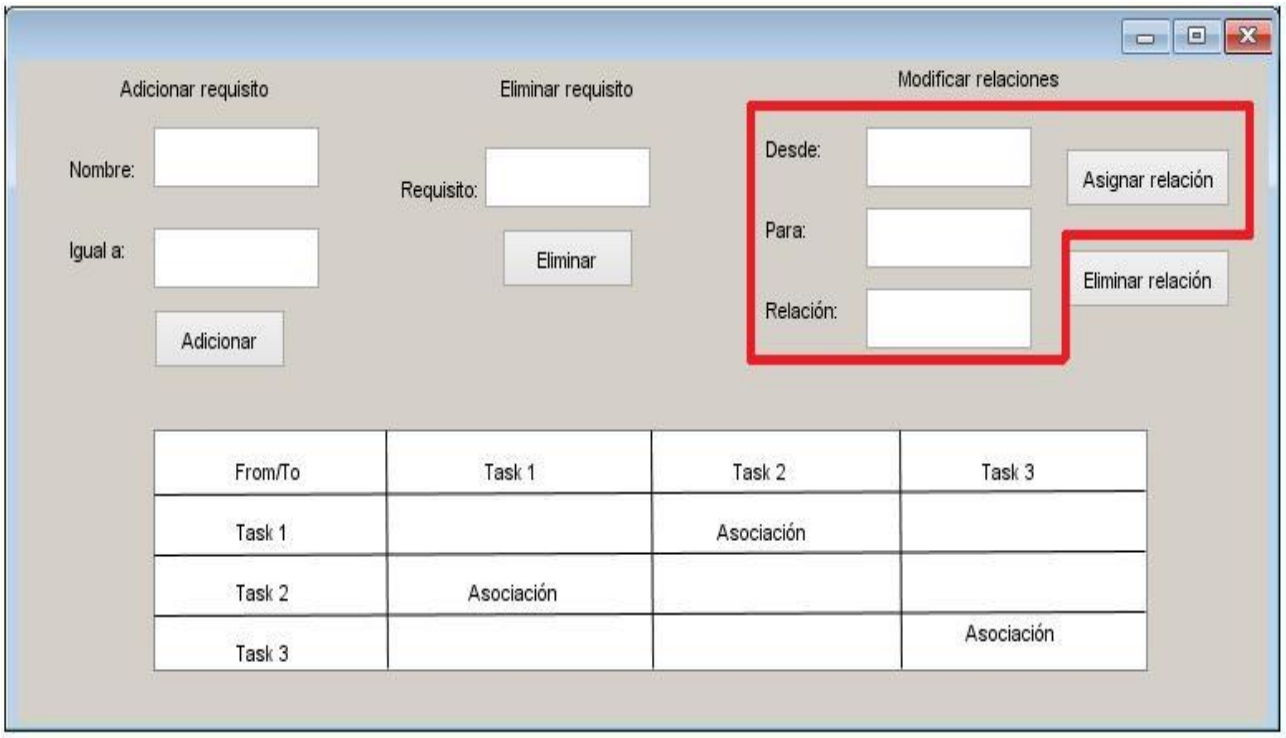


Figura 24.Prototipo del requisito Establecer relaciones entre requisitos

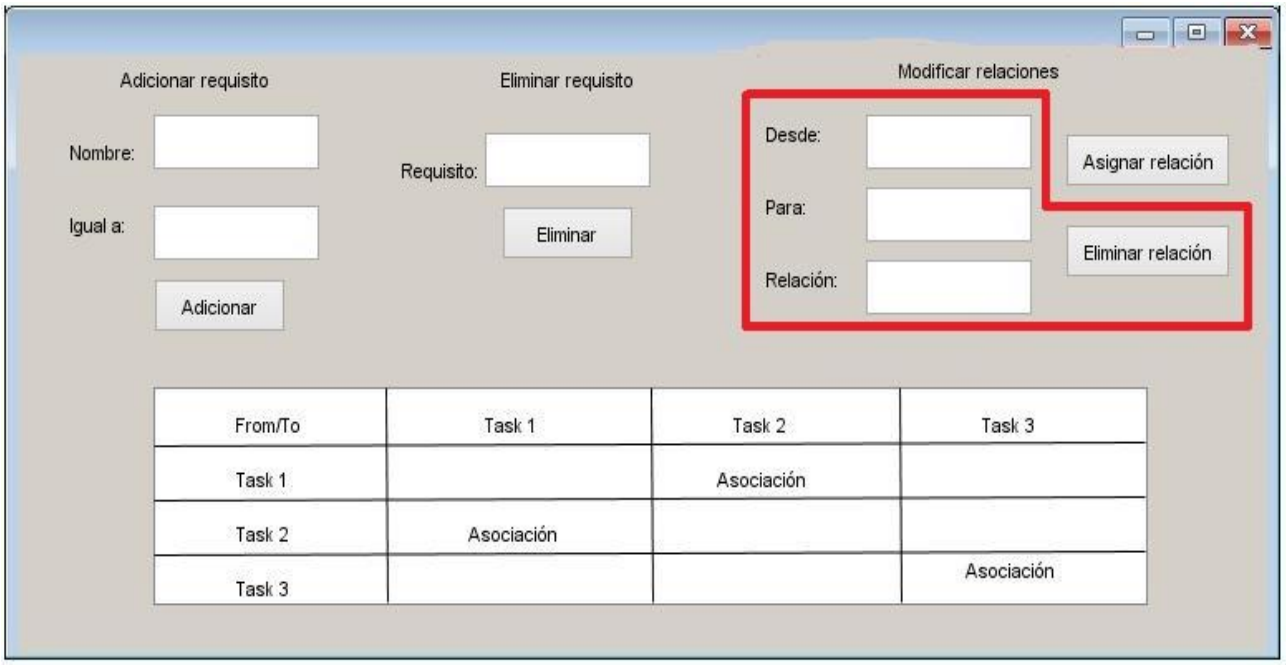


Figura 25.Prototipo del requisito Eliminar relaciones entre requisitos

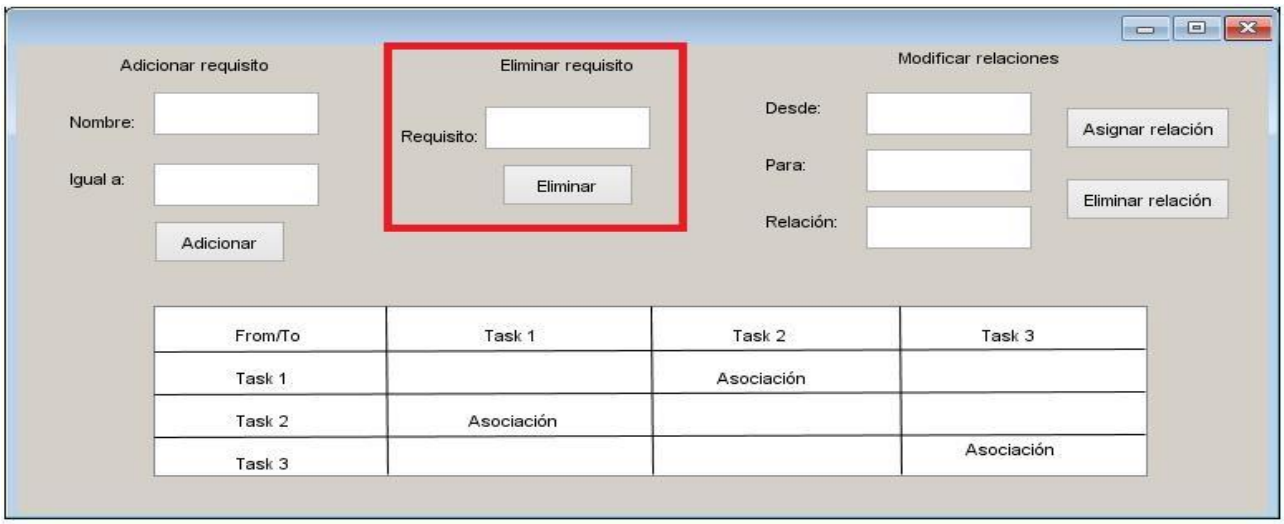


Figura 26. Prototipo del requisito Eliminar requisito

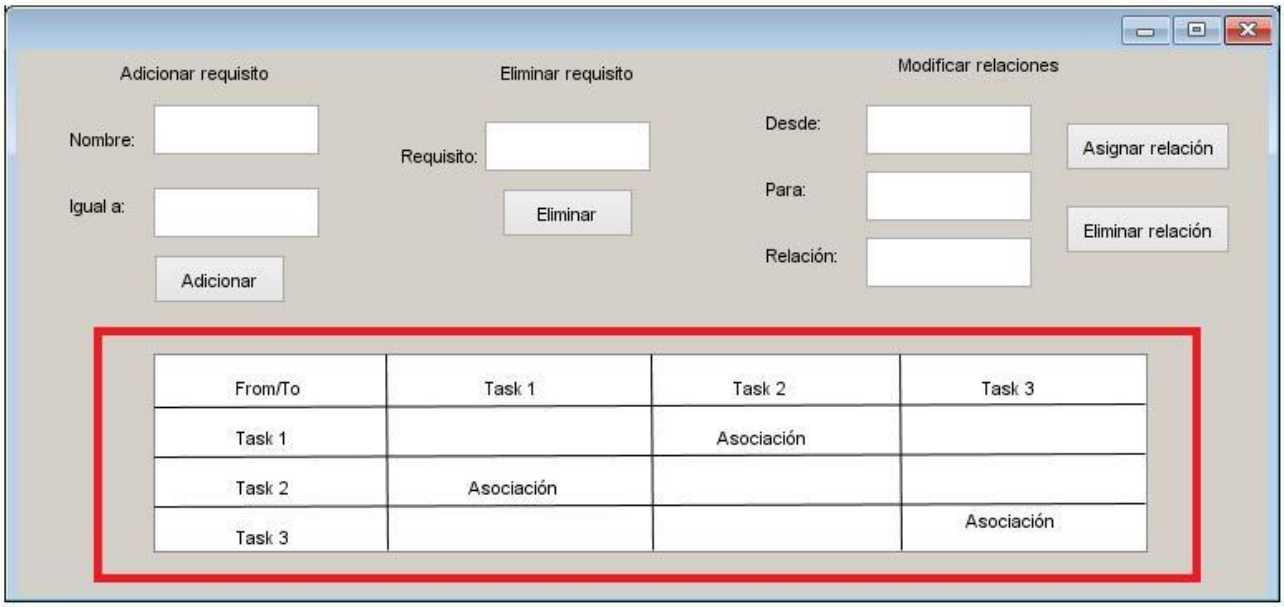


Figura 27. Prototipo del requisito Mostrar relaciones entre requisitos

Anexo #3: Resultado de la aplicación de las métricas TOC y RC a la extensión MN-DR

Figura 28. Resultados de la métrica TOC en MN-DR

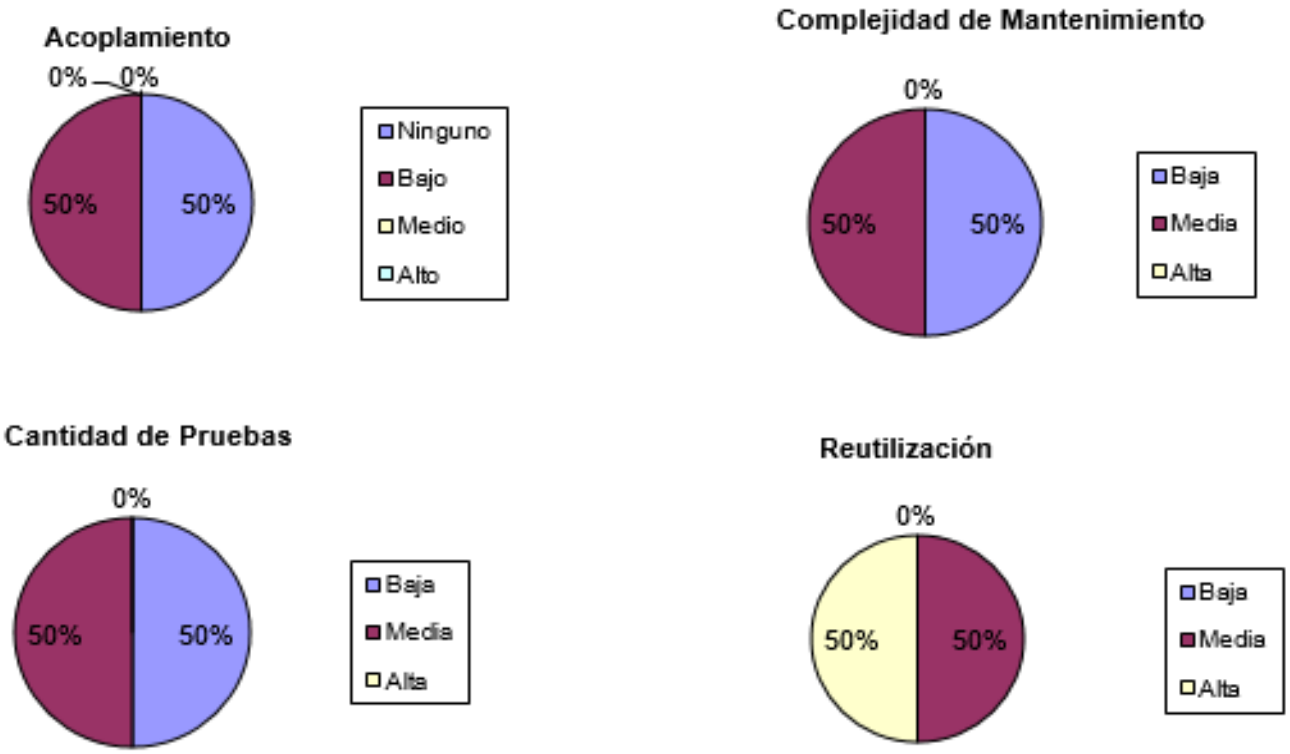


Figura 29. Resultados de la métrica RC en MN-DR

Anexo #4: Acta de liberación de las extensiones DR-PN y MN-DR

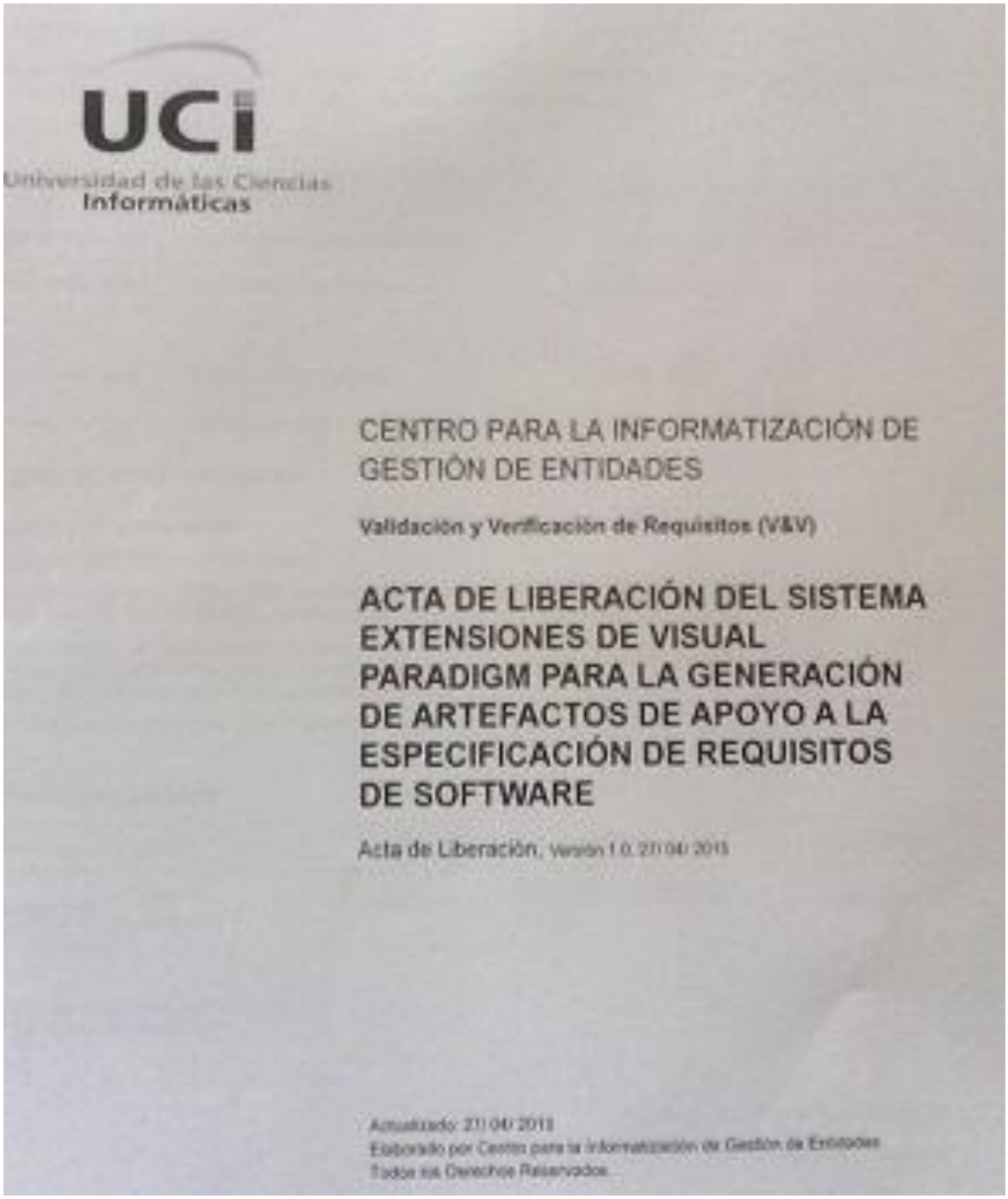


Figura 30. Página principal del documento de liberación de las extensiones

UCI | CIG-AP-CU-00001 | Acta de Liberación

3.1 Cantidad total de horas empleadas y rango de fechas:

Se emplearon un total de 12 horas efectivas de trabajo con la siguiente distribución: 8h en la primera iteración (sucedido el día 15/04/2015) y 4 horas en la 2da iteración (sucedido el día 20/04/2015).

3.2 Estructura del equipo de prueba empleado y turnos de trabajo:

Las pruebas se realizaron en un total de 2 turnos de trabajo, con un 1 probador.

4 Evaluado por:

4.1 Especialista principal Asignado:

Ing. Maylevis Morejón Valdés

4.2 Otro personal especializado participante:

Nombres y Apellidos	Cargo
Ing. Maylevis Morejón Valdés	Administrador de Calidad
Ing. Yisan Antonio López	Probador

Ing. Yisel Rojo Benítez
Asesor de Calidad

Elizabeth Verdecia Martínez
Responsable por el Equipo de Desarrollo

CENTRO DE INFORMATIZACIÓN DE LA GESTIÓN DE ENTIDADES
Universidad de las Ciencias Informáticas
Carretera a San Antonio Km 2 ½, Torrens, Boyeros, Ciudad de La Habana, Cuba
Teléfono: + 53 (7) 837 3660
E-mail: software.gestion@uci.cu

Figura 31. Firmas de los involucrados en el documento de liberación.