Universidad de las Ciencias Informáticas

Facultad 15

“Ingeniería de requisitos de una herramienta para la gestión de requisitos en los proyectos productivos que se desarrollan en la Universidad de las Ciencias Informáticas”.

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

Autores: Arelis Tamayo Carvajal.
Elisabeth Hernández Rosario

Tutor (es): Ing**.** Martín Villalón Cruzata.
Ing. Ariel Morales Malpica.

Ciudad de La Habana, Cuba, 2009-2010

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la facultad 15 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ___ días del mes de junio del año 2010.

Arelis Tamayo Carvajal

_____

(Autor)

Elizabeth Hernández Rosario

_____

(Autor)

Martín Villalón Cruzata

_____

(Tutor)

Ariel Eduardo Morales Malpica

_____

(Tutor)

Agradecimientos

A nuestros tutores Martín y Ariel por su preocupación y por brindarnos su apoyo incondicional en la realización de este trabajo que sin ellos no lo hubiésemos logrado.

A nuestros familiares por agregar cada día un granito de arena en nuestra formación y confiar plenamente en nosotras.

A la universidad (UCI) y a nuestro comandante en jefe Fidel por contribuir en nuestra formación como profesionales y ser parte de este sueño.

A los profesores, compañeros y amigos que de alguna forma ayudaron a que hoy seamos profesionales.

Muchísimas gracias a todos...

Dedicatoria

A mis abuelos que ya no se encuentran por ser la inspiración de este trabajo y de querer este sueño para mí, a ellos se los debo.

A mi padre Joaquín, a mi tía Loyda por estar siempre pendiente de mí y guiándome por el buen camino. Los quiero mucho.

A mi madre que es la razón principal de que yo sea hoy realmente una profesional. La amo.

A mi familia que de alguna forma u otra siempre me han ayudado mucho.

A mi novio Yaimel por darme su amor y apoyo incondicional en los momentos buenos y malos. Te amo.

A mi hermano Manuel, lo quiero muchísimo.

A mi compañera de tesis Elizabeth por dejarme hacer este trabajo con ella y formar parte en mi vida universitaria.

A mis compañeros de grupo por estar en momentos importantes.

Arelis

Resumen

El levantamiento de requisitos es el desarrollo de la actividad necesaria para el inicio de un software. La Universidad de las Ciencias Informáticas (UCI) desarrolla proyectos de software, pero no se gestionan los requisitos de igual manera en sus proyectos; además, las herramientas que utilizan para gestionar los requisitos no cumplen con todas las fases de la ingeniería de requisitos y no son libres. De ahí surge la necesidad de una propuesta para crear una herramienta de gestión de requisitos en función de mejorar la calidad en los proyectos.

El presente trabajo aborda la aplicación de la ingeniería de requisitos a una herramienta para la gestión de requisitos que facilite el trabajo en los proyectos productivos. El propósito trazado consistió en el desarrollo de los artefactos de software que influyen en la transformación de las necesidades de los clientes y usuarios en requisitos de software. Por ende fue necesario analizar, especificar, verificar, validar y gestionar los requisitos identificados. Se determinaron las etapas de la ingeniería de requisitos como elicitación, análisis, especificación, validación y gestión. Para verificar la calidad de la especificación de los requisitos se aplicaron métricas y listas de chequeo, obteniéndose en ambos casos buenos resultados.

PALABRAS CLAVE

Ingeniería de requisitos, artefactos de software, calidad, métricas.

Índice de figuras

Índice de contenido

Introducción

A pesar del vertiginoso desarrollo de software que existe en el mundo, los productos que se obtienen no poseen la calidad que exige el mercado altamente competitivo de estos tiempos. Son demasiadas las exigencias con el avance que existe en el desarrollo de software. Los usuarios y clientes son muy exigentes en cuanto a sus necesidades y a las expectativas que esperan obtener, estas necesidades no son satisfechas en su totalidad, por lo que los productos no llegan a cumplir con sus objetivos y por tal motivo son rechazados. Estos problemas son causados en parte por la mala captura de requisitos, no realizar un buen análisis de los requisitos detectados, la especificación incompleta o mal elaborada, administración insuficiente de requisitos, la comunicación imprecisa entre los presentes que participan en la gestión de los mismos o validaciones tardías de los requisitos.

En el mundo se siguen realizando estudios para encontrar una vía más eficiente de gestionar los requisitos que genera un software. Nuestro país no está ajeno a esta situación y para ello la Universidad de las Ciencias Informáticas (UCI), en su proceso de investigación de desarrollo, busca que la calidad del software que produce sea elevada y su plan de culminación sea en un tiempo aceptable. En la UCI no se gestiona de igual manera los requisitos en cada uno de sus proyectos.

La elicitación de requisitos de los proyectos productivos de la UCI aplica diferentes técnicas de recopilación. Entre ellas están: la entrevista, cuestionarios, revisión de documentos e investigaciones, tomas de decisiones participativas, trabajo creativo en grupos y tormenta de ideas. Seguidamente la información recopilada se procesa a través de las herramientas de modelado para llevarla a un nivel que los desarrolladores sean capaces de entender. Pero sucede que estas herramientas, en su mayoría, son propietarias y el país no está en condiciones de adquirirlas debido a que son muy costosas en el mercado internacional. No solo son costosos los software sino sus licencias también. Otra de las problemáticas es que como son software propietario y la mayoría de estas empresas radican en Estados Unidos, por el actual bloqueo muchas de ellas son limitadas a vender cualquier producto informático a Cuba.

Existen herramientas que son privativas las cuales son muy costosas y existen otras que son libres y multiplataforma pero estas no cubren completamente todas las fases de la gestión de los requisitos. Por lo

que se hace necesario desarrollar una herramienta libre, multiplataforma, que cubra todas las fases de la ingeniería de requisitos y sea accesible para todos los usuarios.

Problema científico

La heterogénea aplicación de herramientas de gestión de requisitos que abarcan de manera parcial las etapas de la ingeniería de requisitos no garantiza una adecuada gestión de los mismos en los proyectos productivos de la Universidad de las Ciencias Informáticas.

Objetivo general

Desarrollar la ingeniería de requisitos de una herramienta que permita la gestión de requisitos en los proyectos productivos de la Universidad de las Ciencias Informáticas.

Objeto de estudio

Proceso de desarrollo de software.

Campo de acción

Ingeniería de requisitos de una herramienta para la gestión de requisitos en el desarrollo de software.

Hipótesis

Si se desarrolla la ingeniería de requisitos de una herramienta para la gestión de requisitos de los proyectos productivos de la Universidad de las Ciencias Informáticas, entonces se facilitará un posterior diseño e implementación.

Tareas de la investigación

1. Realización del marco teórico.
2. Identificación de los requisitos.
3. Especificación del sistema.
4. Validación de la especificación del sistema.
5. Gestión de los requisitos.

Posibles resultados

1. La especificación de requisitos de una herramienta libre multiplataforma que permita llevar a cabo las etapas de la ingeniería de requisitos.
2. Documentar los artefactos ingenieriles propuestos en cada fase y flujo de trabajo.

Métodos y técnicas de investigación a utilizar

Métodos teóricos:

Histórico - Lógico: para el análisis de la trayectoria de la ingeniería de requisitos y para expresar teóricamente sus elementos fundamentales.

Hipotético - Deductivo: para la confección de la hipótesis.

Modelación: para la creación de modelos y diagramas, con el objetivo de explicar cómo funciona el proceso de la herramienta para la gestión de requisitos.

Métodos empíricos:

Entrevista: para interactuar con los clientes e identificar los requisitos del sistema.

Estructura de la tesis

El presente trabajo está conformado por 3 capítulos, los cuales están estructurados de la siguiente manera:

Capítulo 1. Marco Teórico

En este capítulo se hace un estudio de las diferentes metodologías para el desarrollo del software como las herramientas y lenguajes de modelados que se utilizan para modelar los artefactos que proponen estas metodologías. Se abordan las etapas de la ingeniería de requisitos así como los patrones de casos de uso y métricas para facilitar más el trabajo y tener buena calidad.

Capítulo 2. Solución propuesta. Análisis del sistema

En este capítulo se modelan los artefactos necesarios para el negocio y el análisis, utilizando los diagramas de casos de uso del negocio y los diagramas de actividad. Se presenta un conjunto de requisitos tanto funcionales como no funcionales, así como la matriz de trazabilidad. Se muestran también los actores, diagramas de caso de uso del sistema y sus descripciones.

Capítulo 3. Análisis de los resultados

En este capítulo se lleva a cabo la etapa de verificación y validación de requisitos para las primeras etapas (elicitación, análisis y especificación). Se aplican métricas para medir la calidad de los requisitos identificados. Se aplican las técnicas de captura de requisitos conjuntamente con los prototipos de interfaz para conseguir la aceptación por parte de los clientes y usuarios.

Capítulo 1. Marco Teórico

1.1.Introducción.

En el presente capítulo se hace un estudio de las diferentes metodologías, herramientas CASE de desarrollo y lenguajes de modelado más utilizadas en el mundo actual, para llevar un buen análisis. Se exponen los principales patrones existentes para el desarrollo de los casos de uso. Se tratan aspectos importantes de la ingeniería de requisitos y las métricas que favorecen la calidad. Finalmente, se hace una selección de las técnicas y herramientas que se usarán para la realización del análisis de la herramienta de gestión de requisitos.

1.2.Metodologías de desarrollo de software.

Para desarrollar un proyecto de software es necesario establecer un enfoque disciplinado y sistemático. Las metodologías de desarrollo influyen directamente en el proceso de construcción y se elaboran a partir del marco definido por uno o más ciclos de vida (Piattini, 1996). Define metodología como un conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información.

Por lo tanto, una metodología es un conjunto de componentes que especifican:

- ¿Cómo se debe dividir un proyecto en etapas?
- ¿Qué tareas se llevan a cabo en cada etapa?
- ¿Qué salidas se producen y cuándo se deben producir?
- ¿Qué restricciones se aplican?
- ¿Qué herramientas se van a utilizar?
- ¿Cómo se gestiona y controla un proyecto?

Las principales ventajas facilitadas por una metodología para el desarrollo de software son (Sánchez M. A., 2004):

Mejora de los procesos de desarrollo: esto trae consigo que las personas del proyecto trabajen bajo un marco común. Uso de herramientas de ingeniería software y recopilación de mejores prácticas para proyectos futuros.

Mejora de los productos: asegura que los productos cumplan con los objetivos de calidad propuestos, detección temprana de errores y se garantiza la trazabilidad de los productos a lo largo del desarrollo.

Mejora de las relaciones con el cliente: el cliente percibe el orden en los procesos, le facilita al cliente el seguimiento de la evolución del proyecto y se establecen mecanismos para asegurar que los productos desarrollados cumplen con las expectativas del cliente.

Para conseguir que una metodología se aplique adecuadamente es importante que sea fácil de comprender y de aplicar, además de aportar beneficios tangibles a todos los proyectos. Para guiar el desarrollo del software en la actualidad existen varias metodologías como son las ágiles y las tradicionales, entre las ágiles están (Extreme Programming (XP), Scrum), y entre las tradicionales (Rational Unified Process (RUP), Microsoft Solution Framework (MSF)). (Sánchez M. A., 2004).

Seguidamente se muestran algunos aspectos importantes de algunas metodologías.

1.2.1. XP (eXtremePrograming).

XP es una de las metodologías más exitosas existentes en la actualidad, esta se basa en la simplicidad, la comunicación y programación rápida o extrema, es una metodología ligera de desarrollo de software y es considera muy útil para proyectos a corto plazo. (Barbone, 2007).

Entre sus principales objetivos se encuentran los siguientes:

La satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, debemos responder muy rápido a las necesidades del cliente. Incluso cuando los cambios sean al final del ciclo de la programación. XP define 4 variables fundamentales para los proyectos de software (coste, tiempo, calidad y ámbito). (Solís, 2003).

Esta metodología está compuesta por varias fases (ver figura 1):

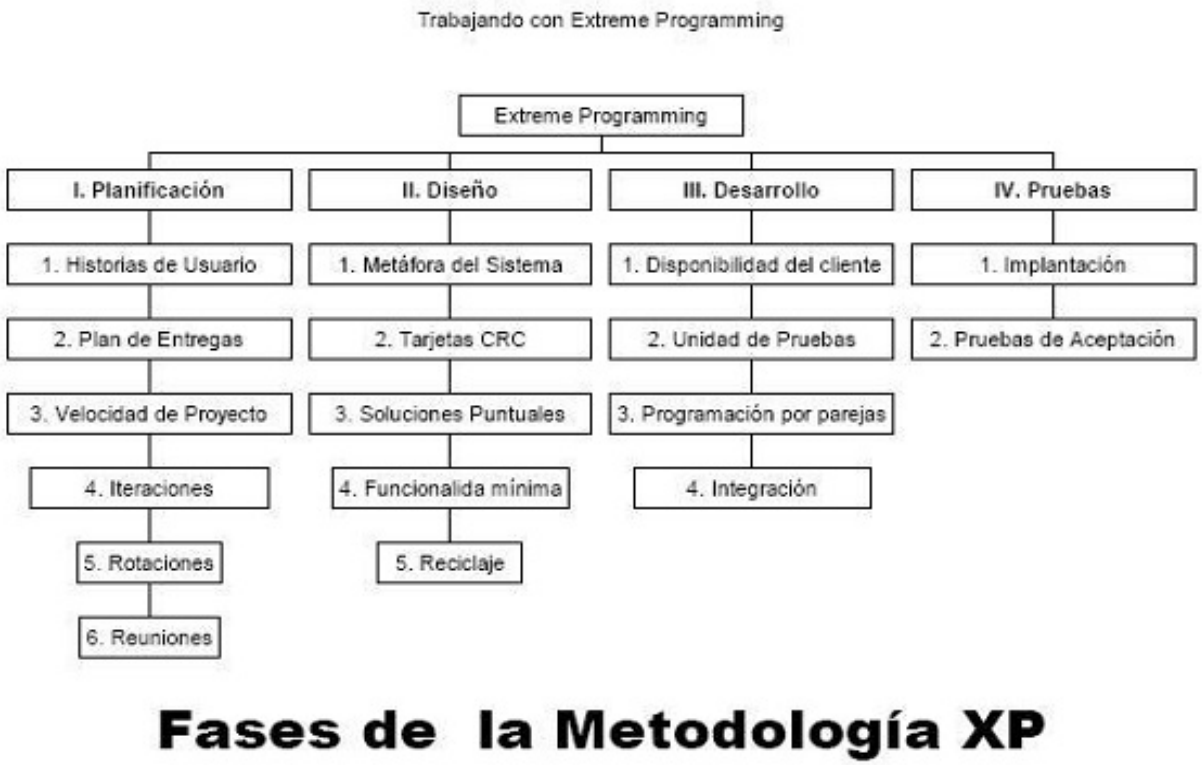


Figura 1. Fases de la Metodología XP

Acerca de esta metodología existen varias críticas de las cuales se muestran algunas a continuación:

Se critica el mito de las 40 horas semanales, y que es un lujo para las exigencias del mercado.

Se dice que solo puede funcionar con programadores muy buenos, que son capaces de hacer un buen diseño, sencillo y fácilmente extensible.

XP es más una filosofía de trabajo que una metodología. Por otro lado, ninguna de las prácticas defendidas por XP son invención de este método, XP lo que hace es ponerlas todas juntas.

XP está diseñado para grupos de pequeños programadores más de 10 ya sería muy complicado, y para que estén en el mismo centro de trabajo.

Las metodologías tradicionales imponen un proceso disciplinado para tratar de hacer el trabajo predecible, eficiente y planificado. Estos métodos están orientados a documentos y se vuelven demasiado burocráticos e ineficaces. XP es más liviana y ágil y está orientada más a las personas que a los procesos. (Solís, 2003.)

1.2.2. Scrum.

Scrum es una de las metodologías ágiles más conocida para la gestión de proyectos. Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Indicada especialmente para proyectos con un rápido cambio en los requisitos. Sus principales característica son:

El desarrollo del software se realiza mediante iteraciones denominadas sprints con una duración de 30 días. Al final de cada sprints se obtiene un producto entregable que se va aumentando en sucesivos sprints, priorizándose aquellos aspectos que contribuyen con mayor funcionalidad y valor al dueño del producto.

Reuniones a largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Scrum es además una metodología indicada para pequeños equipos de desarrollo y se orienta a una entrega rápida de resultados y a una alta flexibilidad. (Acuña, 2009).

Ventajas que proporciona Scrum

Permite a las organizaciones eliminar los impedimentos clásicos en el desarrollo de los proyectos, incrementando la satisfacción de los clientes a través de la realización de entregas frecuentes de resultados tangibles y la integración activamente en el ciclo de desarrollo, lo que permite proporcionar una mayor adaptación a sus necesidades.(Acuña, 2009).

Scrum proporciona la formación de equipos de trabajo autosuficientes, disminuyendo la carga de gestión y facilitando a los miembros del equipo un entorno amigable y productivo para desarrollar sus habilidades al máximo. Este entorno facilita además mayor calidad de vida a los trabajadores y mejora drásticamente la moral en las organizaciones (Acuña, 2009).

1.2.3. Microsoft Solution Framework (MSF).

Esta metodología se basa principalmente en los modelos espiral y cascada (hitos y fases). Como su nombre lo dice fue desarrollado por Microsoft con el objetivo de crear un modelo estructurado basado en una estructura de trabajo en desarrollo de software. Tiene como principios fundamentales la comunicación entre clientes y desarrolladores. MSF es un proceso versionado y se debe crear versiones para el negocio de cada cliente, debe ser ágil, ya que es menos abultado que RUP. (Sánchez M. A., 2004).

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto las cuales son: (Sánchez M. A., 2004).

- Modelo de Equipo.
- Modelo de Proceso.
- Modelo de Gestión del Riesgo.
- Modelo de Diseño del Proceso.
- Modelo de Aplicación.

Ventajas que proporciona utilizar MSF

Entre las principales ventajas que caracterizan a MSF se encuentra que al ser un modelo desarrollado por Microsoft se puede tener mayor soporte y mantenimiento. También es muy adaptable y flexible ya que es utilizado en el ambiente de desarrollo de cualquier cliente. MSF se adapta además a proyectos de cualquier dimensión y cualquier tecnología. (Sánchez M. A., 2004).

Desventajas ocasionadas por MSF

Utilizar esta metodología tiene como principal desventaja tornar un trabajo bastante largo, ya que para cada fase se debe documentar profundamente todo lo que se haga, lo cual es un trabajo muy tedioso a la hora de emplear la misma.(Sánchez M. A., 2004).

1.2.4. Rational Unified Process (RUP).

Es una metodología orientada a procesos, define un ciclo de vida iterativo incremental priorizando el uso de lenguajes de modelado, casos de uso y centrado en la arquitectura. Es un proceso de Ingeniería de software que proporciona una visión disciplinada para la asignación de tareas y responsabilidades en las organizaciones de desarrollo de software. (Jacobson, 2000).

Se caracteriza por ser:

- Iterativo e incremental: cada fase se desarrolla en iteraciones. Se divide el trabajo en partes más pequeñas o mini-proyectos, donde cada uno es una iteración que resulta en un incremento.
- Centrado en la arquitectura: la arquitectura describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.
- Guiado por los casos de uso: los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. (Jacobson, 2000)

El modelo RUP está compuesto por:

- Roles.
- Actividades.
- Artefactos.
- Disciplina.
- Flujos de trabajo.

RUP divide el proceso en 4 fases y flujos de trabajo (ver figura 2), dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

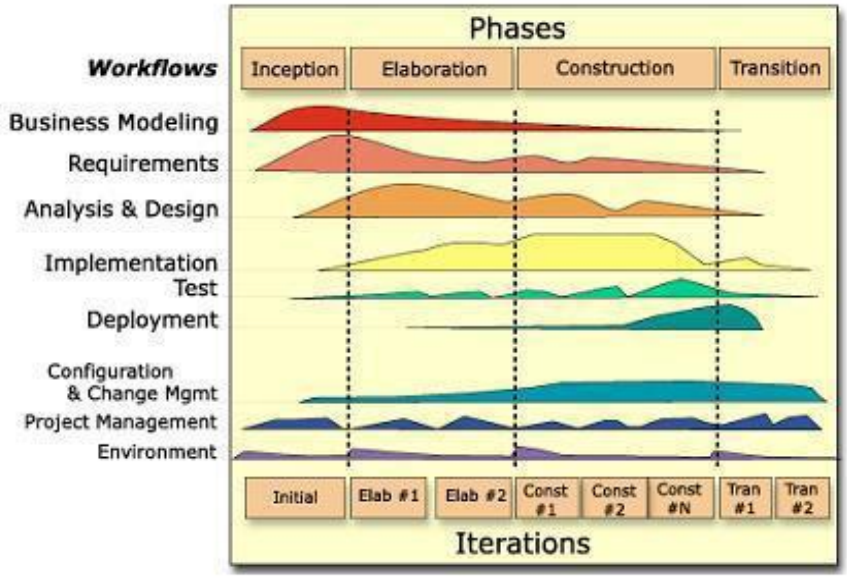


Figura 2. Proceso unificado de desarrollo de software (fase y flujo de trabajo)

Fases e iteraciones que propone RUP

Fases que dividen el ciclo de vida de un producto RUP:

En las iteraciones de cada fase se hacen diferentes esfuerzos en diferentes actividades, las cuales concluyen con un hito bien definido donde deben tomarse ciertas decisiones (Sánchez, 2004):

Inicio: es la fase de la idea, de la visión inicial del producto, su alcance. Se identifican los principales casos de uso y se identifican los riesgos.

Elaboración: comprende la planificación de las actividades y del equipo necesario. Se hace un plan del proyecto, se completan los casos de uso y se eliminan los riesgos. Termina con el “hito de Arquitectura”.

Construcción: se concentra en la obtención de un producto totalmente operativo y eficiente. Termina con el “hito del inicio de la capacidad operativa”.

Transición: traspaso del producto a los usuarios. Incluye: manufactura, envío, formación, asistencia y el mantenimiento hasta lograr la satisfacción de los usuarios.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. (Sánchez, 2004).

Controlar los cambios al software:

La capacidad de administrar los cambios es esencial en ambientes en los cuales el cambio es inevitable. RUP describe cómo controlar, rastrear y monitorear los cambios para permitir un desarrollo iterativo exitoso. Es también una guía para establecer espacios de trabajo seguros para cada desarrollador, suministrando el aislamiento de los cambios hechos en otros espacios de trabajo y controlando los cambios de todos los elementos de software (modelos, códigos y documentos). Describe cómo automatizar la integración y administrar la conformación de raleases.

RUP se propone varias metas como:

- Reducir en gran medida el riesgo que representa la construcción de sistemas complejos, porque evoluciona de forma incremental partiendo de sistemas más pequeños en los que ya se tiene confianza.
- Proveer un camino metódico, sistemático para desarrollar, diseñar y validar una arquitectura.
- Proporcionar un enfoque disciplinado para asignar tareas y responsabilidades dentro del desarrollo del sistema.
- Asegurar la producción de un software de alta calidad que reúna las necesidades de los usuarios finales dentro de un plan y un presupuesto predecible.

Por las características y ventajas que tiene esta metodología se decidió utilizarla para el análisis del software, ya que es una de las más aplicadas en la actualidad, por ser un proceso iterativo e incremental que permite la realización de futuras versiones de la herramienta, además es flexible, posibilita dividir el trabajo en fases teniendo bien definidas las tareas a realizar en cada una de ellas, establece roles de trabajo dentro del proyecto con el fin de implantar una organización del equipo de trabajo, asignando

tareas y responsabilidades a cada uno de sus miembros. Dirige las tareas de cada desarrollador por separado, alcanzando así un mejor funcionamiento del proyecto, ofrece criterios para el control, medición de productos y actividades, obteniendo un sistema con mayor calidad. Brinda una amplia documentación facilitando su uso. Esta metodología al estar basada en una fuerte interacción con los clientes y usuarios, permite obtener productos adecuados a las necesidades reales ahorrando esfuerzos y aumentando la satisfacción final del usuario.

1.3.Herramientas de desarrollo de software.

Las herramientas CASE son un conjunto de métodos, utilidades y técnicas que facilitan la informatización del ciclo de vida del desarrollo de sistema de información, completamente o en alguna de sus fases.

Rational Rose

Es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto y provee un lenguaje de modelado común para el equipo, permitiendo rápidamente la creación de un software de calidad.

Entre sus principales características se encuentran:
(Cutisaca, 2008)

- Realiza chequeo semántico de los modelos.
- Desarrollo multiusuario.
- Generación de documentación automáticamente.
- Generación de código a partir de los Modelos.
- Posee Ingeniería Inversa (crear modelo a partir de código).

Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Entre sus principales ventajas se encuentran:

- Utiliza la notación estándar en la arquitectura de software (UML), la cual permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común.
- Los diseñadores pueden modelar sus componentes e interfaces de forma individual y luego unirlos con otros componentes del proyecto.
- Rose soporta la construcción de componentes en lenguajes como C++, Visual Basic, Java y Ada.

Entre sus principales desventajas se encuentran:

- Poco amigable a los usuarios.
- Posee una instalación costosa.
- Alta capacidad de procesamiento.

Rational ofrece un Proceso Unificado (RUP) para el desarrollo de los proyectos de software, desde la etapa de Ingeniería de Requisitos hasta la etapa de pruebas. Para cada una de estas etapas existe una herramienta que ayuda en la administración de los proyectos. Rose es la herramienta de Rational para la etapa de análisis y diseño de sistemas. Es considerada la herramienta de análisis, diseño, modelado y construcción de software orientado a objetos líder en el mercado. (Cutisaca, 2008)

Enterprise Architect

Enterprise Architect es una herramienta completa de análisis y diseño UML, que cubre el desarrollo de software desde la concepción de las exigencias a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. Enterprise Architect (EA) es una herramienta gráfica multiusuario basada en Windows, diseñada para ayudarle a construir un software robusto y conservable. Además despliega documentación de salida flexible y de alta calidad.

Entre sus principales ventajas se encuentran:

Velocidad, estabilidad y buen rendimiento: el Lenguaje Unificado de Modelado provee beneficios significativos para ayudar a construir modelos de sistemas de software rigurosos y donde es posible mantener la trazabilidad de manera consistente. EA soporta este proceso en un ambiente fácil de usar, rápido y flexible.

Trazabilidad de extremo a extremo: EA provee trazabilidad completa desde el análisis de requisitos hasta los artefactos de análisis y diseño a través de la implementación y el despliegue. Combinados con la ubicación de recursos y tareas incorporados, los equipos de administradores de proyectos y calidad están equipados con la información que ellos necesitan para ayudarles a entregar proyectos en tiempo.

Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Permite realizar los prototipos no funcionales de la aplicación.

Visual Paradigm ofrece:

- Entorno de creación de diagramas para UML 2.0.
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad en múltiples plataformas.

Se integra con herramientas Java como:

- Eclipse.
- JBuilder.
- NetBeans/Sun ONE.
- WeblogicWorkshop.
- JDeveloper.

- IntelliJ IDE.

Se decidió utilizar Visual Paradigm ya que brinda la posibilidad de organizar los diagramas y la documentación asociada al desarrollo de la herramienta, ofrece un diseño centrado en casos de uso; además de que permite la disponibilidad de múltiples versiones futuras para la herramienta y en múltiples plataformas, genera documentación y posee abundantes tutoriales de UML que facilitan una mejor comprensión del mismo. Esta herramienta posibilita la realización de los prototipos no funcionales de la aplicación permitiendo validar el sistema y soporta el ciclo de vida completo del desarrollo de software.

1.4.Lenguaje de modelado.

Los lenguajes de modelado permiten la representación del comportamiento y la estructura de los sistemas del mundo real.

IDEF0

IDEF0 es un método diseñado para modelar decisiones, acciones y actividades de una organización o sistema. Los efectivos modelos de IDEF0 permiten organizar el análisis de un sistema y facilita la comunicación entre el analista y el cliente. Como herramienta de análisis, IDEF0 asiste al analista para identificar que funciones se llevan a cabo, lo que se necesita para realizarlas, lo que se hace bien y mal en los procesos que se modelan.

IDEF0 tiene las características siguientes:

- Es comprensivo, expresivo, capaz gráficamente de representar una variedad amplia del negocio y de otros tipos de operaciones de la empresa a cualquier nivel del detalle.
- Es un lenguaje coherente, simple, que prevé la expresión rigurosa, exacta, y promueve la consistencia del uso y de la interpretación.

Puede ser generado por una gran variedad de herramientas gráficas en computadores. Muchos productos comerciales apoyan específicamente el desarrollo y el análisis de los diagramas IDEF0 y de sus modelos.

UML (Lenguaje Unificado de Modelado).

Los lenguajes de modelado permiten la representación del comportamiento y la estructura de los sistemas del mundo real. El lenguaje unificado de modelado (UML) es usado para especificar, visualizar y documentar los diferentes aspectos relativos a un sistema de software bajo desarrollo, así como para modelado de negocios y almacenamiento de datos. (Guerrero, 2009)(Zapata, 2008).

Por lo que podemos decir que UML se identifica con las siguientes características:

- Divide un proyecto en un número de diagramas que representan las distintas vistas del proyecto y juntos representan la arquitectura del mismo.
- Permite describir un sistema en diferentes niveles de abstracción, simplificando la complejidad sin perder información, para que los usuarios y desarrolladores comprendan las características de la aplicación.
- Se quiere convertir en un lenguaje estándar con el que sea posible modelar todos los componentes del desarrollo de una aplicación, sin embargo no pretende definir un modelo de desarrollo sino únicamente un lenguaje de modelado.

UML se crea con el objetivo de crear sistemas de modelado que utilizan conceptos orientados a objetos. Es importante destacar que UML está constituido por diagramas y brinda características que permiten el entendimiento y organización de los mismos, se clasifican en estructura estática, de comportamiento y los de implementación.

Diagramas de estructura estática

- Diagramas de casos de uso.
- Diagrama de clases.
- Diagrama de objetos.

Diagramas de comportamiento

- Diagrama de interacción.
- Diagrama de secuencia.

- Diagrama de colaboración.
- Diagrama de actividad.
- Diagrama de estados.

Diagramas de implementación

- Diagrama de componentes.
- Diagrama de despliegue.

Un modelo UML identifica lo que supuestamente hará el sistema, y no con seguridad como lo hará. Pero eso no quita que sea una guía y una representación de las especificaciones del sistema. Permitiendo poder tomar decisiones que cumplan con las expectativas de los clientes y los usuarios.

¿Por qué UML?

La elección del lenguaje para modelar depende de muchos factores. UML es una notación, es decir, una serie de reglas y recomendaciones para representar modelos. Permite documentar y especificar los elementos creados mediante un lenguaje común describiendo modelos. UML surge como respuesta al primer problema identificado para contar con un lenguaje estándar y poder escribir planos de software. Además de que RUP propone como lenguaje de modelado a UML. Posibilita a los clientes y otras personas involucradas en el desarrollo del sistema tener un entendimiento común de la herramienta, lo cual es de vital importancia para el desarrollo eficiente del sistema. Brinda la posibilidad de documentar, especificar, construir y visualizar el sistema. (Guerrero, 2009)(Zapata, 2008)

1.5.Ingeniería de requisitos.

La ingeniería de requisitos (IR), disciplina que inicia el proceso de desarrollo de un software, tiene como principal objetivo la especificación de requisitos que cumpla con las necesidades del cliente, aquí se identifican dos procesos fundamentales, el primero, cuál es el propósito del sistema que se va a desarrollar, y el segundo el contexto en que será usado. (Gallego, 2006)

La IR es una guía para poder lograr un enlace entre las necesidades de los clientes, usuarios y demás que estén involucrados en el sistema. Consiste en agrupar un conjunto de actividades y transformarlas en requisitos para llevarlas a una descripción completa, precisa y documentada.

Un requisito se puede definir como atributo necesario dentro de un sistema, que puede representar una capacidad, una característica, o un factor de calidad del sistema, de tal manera que sea útil para los clientes o usuarios finales. Como disciplina esta se agrupa por etapas, Elicitación, Análisis, Especificación, Validación y Gestión de Requisitos como principales etapas de la Ingeniería de Requisitos, las cuales se especifican a continuación. (Gallego, 2006)

Etapas de la Ingeniería de Requisitos

Elicitación:

Es el proceso en el cual los desarrolladores y clientes de un sistema software, se comunican, describen, revisan y entienden las necesidades del sistema software y las restricciones que se establecen en el desarrollo del software.

Para obtener resultados esperados por la aplicación se hace necesario aplicar las siguientes técnicas:

Entrevistas:

Es la técnica de elicitación más utilizada. Prácticamente son inevitables a la hora de desarrollar un proyecto ya que es la forma más natural de tener comunicación con los clientes. En las entrevistas se pueden identificar tres fases (preparación, realización y análisis). Para la preparación de la entrevista se debe tener en cuenta el estudio del dominio del problema que no es más que conocer las categorías y conceptos de la sociedad del cliente y usuarios, es necesaria para poder entender dicha sociedad. Seleccionar a la persona a la que se va entrevistar es un punto muy importante para dicha preparación al igual que determinar el objetivo, contenido de la entrevista y su planificación. Para su realización se tiene en cuenta tres puntos importantes que son apertura, desarrollo y terminación. (Toro D. A., septiembre 2000)

Joint Application Development (JAD)

Esta técnica desarrollada por IBM en 1977, es una alternativa a las entrevistas individuales que se desarrollan en unos conjuntos de reuniones en grupo en un plazo de 2 a 4 días, es aquí en estas reuniones donde se ayudan a los usuarios y clientes a formular problemas y llegar a posibles soluciones.
Esta técnica presenta las siguientes ventajas en comparación con las entrevistas individuales:

- Implica más a los clientes y usuarios en el desarrollo.
- Todo el grupo, incluyendo los clientes y los futuros usuarios, revisan la documentación generada, no solo los ingenieros de requisitos.
- Ahorra tiempo al evitar que las opiniones de los clientes se contrasten por separado.

Se distinguen tres fases adaptación, celebración de las secciones JAD y conclusión. (Toro D. A., septiembre 2000)

Tormenta de ideas:

Es una técnica de reuniones en grupo donde su principal objetivo es desarrollar ideas en un ambiente libre de críticas y juicios, estas suelen estar formadas por un grupo de 4 a 10 participantes, donde uno es el jefe, que es el encargado de iniciar más la sección que de controlarla. Como técnica de la elicitación ayuda a generar una gran variedad de vistas del problema y a formularlo de diferentes formas sobre todo al principio cuando los requisitos son muy dudosos. Consta de cuatro fases (preparación, generación, consolidación, y documentación). (Toro D. A., septiembre 2000).

Casos de uso:

Los casos de uso son una técnica para la especificación de requisitos funcionales. Un caso de uso no es más que la descripción de una secuencia de iteraciones entre el sistema y uno o más actores en la que se considera el sistema como una caja negra y en la que los actores obtienen resultados observables. Además pueden servir de base a la propuesta del sistema y a la documentación para los usuarios. (Toro D. A., septiembre 2000).

Artefactos que se generan:

De entrada:

- Documento visión.
- Cronograma.
- Guía de preguntas.

De salida:

- Minuta de entrega.
- Base de documentación.
- Glosario de términos.
- Guía de preguntas (Actualizada).
- Cronograma (Actualizado).
- Documento de los procesos del negocio.
- Documento de revisión técnica formal.
- Listado de requisitos funcionales.
- Lista de riesgos.
- Listas de chequeo.
- Informe de no conformidades.
- Documento de los procesos del negocio (Validado).
- Glosario de términos (Validado).
- Minuta de aceptación.

Análisis:

En esta etapa se estudian los requisitos obtenidos de la etapa anterior, donde se detectan entre otros la presencia de áreas no especializadas, contradicción entre los requisitos y peticiones irrelevantes. El resultado de estas tareas conlleva más de una vez regresar a la primera etapa, para eliminar las inconsistencias y fallas que se han detectado. (Ariza, agosto 2009).

Objetivo: lograr un análisis, negociación de los requisitos para identificar posibles conflictos, caracterizar, priorizar cada requisito y llegar a un completo desarrollo para su especificación.

Actividades a desarrollar:

- Análisis y negociación de los requisitos: analizar los requisitos identificados en la etapa de elicitación y negociar con el cliente nuevos requisitos.
- Revisión técnica formal: realizar una revisión interna del documento de listado de requisitos funcionales para asegurar la calidad del mismo.
- Actualizar cronograma: se actualizan las tareas planificadas en el cronograma.
- Priorizar los requisitos: se priorizan los requisitos identificados de acuerdo con las necesidades de los usuarios ya sea en alto, medio o bajo. Se realiza una negociación con el cliente si existe algún requisito nuevo y se vuelve a definir.

Artefactos que se generan:

De entrada:

- Glosario de términos.
- Cronograma.
- Documento de los procesos del negocio.
- Listado de requisitos funcionales.
- Lista de riesgos.

De salida:

- Listado de requisitos funcionales (Actualizado/Modificado).
- Informe de no conformidades.
- Registro de revisiones.
- Cronograma (Actualizado).

Especificación:

Esta etapa parte de lo elaborado en la etapa anterior, como son las funciones, datos, requisitos no funcionales, objetivos, restricciones de diseño e implementación. Es aquí donde se describen los requisitos, si se tiene dificultades para especificar se debe regresar a la etapa anterior que se crea conveniente. (Ariza, agosto 2009)

Objetivos: lograr la especificación de cada uno de los requisitos capturados, el modelo de casos de uso del sistema así como también la obtención de los prototipos no funcionales.

Actividades a desarrollar:

- Documento de especificación de los requisitos de software: se realiza la documentación en limpio de todos los requisitos que desea el cliente que tenga su sistema a desarrollar.
- Revisión formal interna: se realiza una revisión interna al documento de especificación de requisitos para asegurar la calidad del mismo.
- Revisión con el cliente: realizar una revisión con el cliente de la documentación de especificación de los requisitos de software realizada por los analistas del sistema del equipo de desarrollo.
- Actualizar cronograma: se actualizan las tareas planificadas en el cronograma de la especificación de los requisitos de software.
- Modelo de caso de uso del sistema: luego de terminada la especificación, se seleccionan los casos de usos a implementar, y se documentan.
- Revisión Formal Interna: se realiza una revisión interna al modelo de caso de uso de sistema para asegurar la calidad del mismo.
- Revisión con el cliente: realizar una revisión con el cliente del documento de modelación de los casos de uso del sistema realizado por los analistas del sistema del equipo de desarrollo.
- Actualizar cronograma: se actualizan las tareas planificadas en el cronograma de la modelación de los casos de uso del sistema.

Artefactos que se generan:

De entrada:

- Listado de requisitos funcionales.
- Glosario de términos.
- Cronograma.

De salida:

- Documento de especificación de requisitos de software.
- Documento de especificación de requisitos de software (Aprobado).
- Glosario de términos (Actualizado/Aprobado).
- Cronograma (Actualizado).
- Modelo de caso de usos del sistema.
- Modelo de caso de usos del sistema (Aprobado).
- Registro de revisiones.
- Informe de no conformidades.
- Minuta de aceptación.

Verificación y validación:

Esta etapa realiza la validación e integración final de lo obtenido de las etapas anteriores dando como resultado final el documento de requisitos, en este documento como mínimo existen dos partes que son isométricas entre sí, uno destinado al cliente y el otro técnico orientado a nutrir las restantes etapas de la Ingeniería de Software. (Ariza, agosto de 2009)

En el glosario de la IEEE de 1990 aparecen ambos términos en una única entrada:

Verificación y validación (1): el proceso de determinar si los requisitos para un sistema o componente son completos y correctos, los productos de cada fase de desarrollo satisfacen los requisitos o

condiciones impuestas por la fase previa y el sistema o componente final es acorde con los requisitos especificados.

Curiosamente más adelante en el mismo glosario aparecen también los términos, pero como entradas separadas y no coincide completamente con la definición anterior. En este caso obvian por completo los aspectos relativos a los requisitos:

Verificación (2): (a) el proceso de evaluar un sistema o componente para determinar si los productos de una fase de desarrollo dada satisfacen las condiciones impuestas al comienzo de la fase. (b) prueba formal de la corrección de un programa.

Validación (2): el proceso de evaluar un sistema o componente durante o al final del proceso de desarrollo para determinar si satisface los requisitos especificados. Otras definiciones menos formales se pueden encontrar de la siguiente forma:

Verificación (3): ¿se está construyendo correctamente el producto?

Validación (3): ¿se está construyendo el producto correcto?

En este trabajo se tratan los términos atribuyéndoles significados distintos especialmente en la Ingeniería de Requisitos que no tienen que ir necesariamente unidos.

Por verificación se entenderá el conjunto de actividades relacionadas con la calidad de las especificaciones de requisitos respecto a un conjunto de propiedades deseables.

Se entenderá por validación de requisitos el conjunto de actividades encaminadas a llegar a un acuerdo entre todos los participantes, en el que se ratifique que los requisitos elicitados y analizados representan realmente las necesidades de clientes y usuarios.

Objetivos: lograr la validación de la especificación de requisitos de software y el modelo de casos de uso del sistema.

Actividades a desarrollar:

- Validación de la especificación de los requisitos de software: se realiza una validación en conjunto con el cliente de la especificación de los requisitos de software.
- Validación del modelo de casos de uso del sistema: se realiza una validación en conjunto con el cliente del modelo de casos de uso del sistema.

Artefactos que se generan:

De entrada:

- Documentos de especificación de requisitos de software.
- Modelo de caso de usos del sistema.
- Glosario de términos.

De salida:

- Modelo de caso de uso del sistema (Aprobado).
- Documento de especificación de los requisitos de software (Aprobado).
- Registro de revisiones.
- Informe de no conformidades.
- Glosario de términos (Aprobado).
- Minuta de aceptación.

Técnicas utilizadas para validar los requisitos:

Walkthrough: técnica de análisis estático en la que un programador o diseñador dirige a miembros del equipo de desarrollo u otras personas interesadas a través de un segmento de documentación o código y los participantes realizan comentarios sobre posibles errores y violaciones de estándares de desarrollo.

Los objetivos de una sesión de walkthrough son encontrar conflictos (defectos, omisiones y contradicciones) en el producto que se revisa, de forma que puedan plantearse alternativas y los participantes aumenten su conocimiento sobre el producto en cuestión.

Durante las sesiones de walkthrough, el autor del producto recorre el producto a revisar en detalle permitiendo que los participantes pongan de manifiesto sus opiniones sobre el mismo.

Prototipos

Para validar los requerimientos hallados, se construyen prototipos. Los prototipos son simulaciones del posible producto, que luego son utilizados por el usuario final, permitiéndonos conseguir una importante retroalimentación en cuanto a si el sistema diseñado en base a los requisitos recolectados le permite al usuario realizar su trabajo de manera eficiente y efectiva.

El desarrollo del prototipo comienza con la captura de requisitos. Desarrolladores y clientes se reúnen y definen los objetivos globales del software, identifican todos los requisitos que son conocidos y señalan áreas en las que será necesaria la profundización en las definiciones. Luego, tiene lugar un “diseño rápido” que se centra en una representación de aquellos aspectos del software que serán visibles al usuario (por ejemplo, entradas y formatos de las salidas) y que conlleva a la construcción de un prototipo.

Gestión de requisitos:

Durante el desarrollo del sistema los cambios son inevitables por lo que estos deben pasar por un proceso de gestión. “La gestión de requisitos es un conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar, seguir los requisitos y los cambios en cualquier momento”. (Pressman, 2005).

Luego de identificar los requisitos se puede hacer una gestión de los mismos para luego llevar un control de estos a través de las matrices de trazabilidad, entre las que se encuentran:

- Matriz de seguimiento de características.
- Matriz de seguimiento de orígenes.
- Matriz de seguimiento de dependencias.

- Matriz de seguimiento de subsistemas.
- Matriz de seguimiento de interfaces.

Según (Pressman, 2005), el control de cambios de los requisitos es importante para un buen desarrollo del sistema por lo que recomienda también actividades de la gestión de configuración del software que son muy útiles para la gestión de requisitos.

Herramientas para la gestión de requisitos

Seguidamente se presenta un análisis detallado de 9 herramientas utilizadas para la gestión de requisitos y una comparación entre ellas (ver tabla 1) que se encuentran disponibles en el mercado, basándose en las necesidades en los proyectos productivos de la Universidad de las Ciencias Informáticas, donde se realiza un estudio basándose tanto en las libres como en las privadas, estas herramientas son: OSRMT(Open Source Requirements Management Tool) , REM(REquirements Management), JRequisite, OSEE (Open System Engineering Environment), Caliber, RequisitePro, GatherSpace, IrqA y Raquest.

Tabla 1. Comparación entre las 9 herramientas elegidas para estudio.

Caracterís-ticas	OSMRT	REM	JRequisite	OSEE	Caliber	Requisite Pro	Gather Space	IrqA	RaQuest
Código abierto	x		x	x					
Manejo de trazabilidad	x	x	x	x	x	x	x	x	x
Generación de reportes	x	x			x		x	x	
Documentación	x				x	x	x		
Soporte			x(No es gratuito)		x	x (No es gratuito)	x		

Ciclo de vida completo	x			x					
Control de versiones	x	x	x		x				
Posibilidad de enlace con otras herramientas				x	x	x			x
Manejo de base de datos	x	x	x	x	x	x	x	x	x
Plataforma	Independiente	Windows	Windows	Independiente	Windows	Windows	Independiente	Windows	Windows
Nivel de seguridad	Configurable	Configurable	Configurable	-	-	si	Predeterminados		
Control de inconsistencia					x	x			
Análisis de cambio	x	x			x	x			

Herramientas que son de código abierto (Open Source):

OSRMT (Open SourceRequirements Management Tool): es una herramienta libre desarrollada en Java, utilizada para la gestión de requisitos.

Ventajas y desventajas:

- Permite la gestión de muchos requisitos (casos de uso, requisitos, casos de prueba) y garantiza la trazabilidad entre los mismos.
- Permite llevar un control de quién y cuándo ha realizado un cambio sobre la documentación.
- Permite almacenar direcciones URL y ficheros binarios.
- Permite llevar con control de versiones en los documentos de trabajos.
- Permite definir los niveles de accesibilidad de los usuarios al sistema.
- Realiza búsquedas a través de aplicaciones de filtros.
- Importa y exporta información en XML y HTML.
- Genera reportes de la forma HTML y PDF.
- Tiene un cliente web.
- Visualiza los requisitos de forma jerárquica.
- Es multiplataforma.
- Documentación pobre.
- No genera automáticamente un documento de requisitos para entregar al cliente.
- La interfaz de usuario en ocasiones es poco lenta.

REM (REquirements Management):

Herramienta gratuita diseñada para la gestión de requisitos por la Universidad de Sevilla.

Ventajas y desventajas:

- La visualización de los requisitos es jerárquica y fácil de manejar.
- Se van generando documentos HTM una vez que se introduce la información en la aplicación.
- Consta de una interfaz intuitiva.
- Es gratuita pero no es de código abierto.
- Tiene algunos errores y no consta de ningún soporte.
- Requiere internet Explorer 6.0 y solo funciona bajo plataforma Windows.
- No consta con integralidad web.

- No es multiusuario.
- Facilidad de uso e interfaz aparentemente amigable.
- Solo puede utilizarse para fines académicos.

JRequisite:

Es una herramienta de código abierto desarrollado para la gestión de requisitos puesto en marcha solo la funcionalidad de especificación de caso de uso mediante diagramas de flujo. Es un plugins de eclipse que se basa en el GMS (Framework de Modelado Gráfico).

Ventajas y desventajas:

- Es multiplataforma.
- Permite realizar diagramas de casos de uso por lo que su funcionalidad es muy pobre.
- No tiene manual de usuario.

OSEE (Open System Engineering Environment):

Es una herramienta de código abierto, diseñado para proporcionar un grupo de funcionalidades en una sola herramienta, llevando consigo la gestión de configuración, gestión de requisitos, verificación, validación y gestión de proyecto.

Ventajas y desventajas:

- La documentación es pobre.
- No es muy amigable para el usuario.

Herramientas privadas

Caliber:

Es un repositorio central donde se almacenan todos los requisitos de todos los proyectos.

Ventajas y desventajas:

- Permite visualizar y hacer trazas, con el fin de ver los impactos que traería algún cambio en el desarrollo de un proyecto.
- Permite la conexión a través de distintos tipos de clientes como web, eclipse, visual estudio y Windows.
- Hay que pagar licencia para poder tener la aplicación.
- Interfaz amigable y fácil de usar.
- Posibilidad de ampliación de funcionalidad de la herramienta (API basadas en componentes COM y Java).
- Control de acceso de grupos y usuario.
- Control de versiones, adquiriéndolas mediante compra.
- Historia en los cambios de requisitos de manera automática.
- Identificación de inconsistencia de forma gráfica mediante matriz de trazabilidad.
- No soporta la reutilización y selección de requisitos, solo copiar y pegar dentro y entre proyectos.(Reyes, 2005)

Rational RequisitePro:

Esta herramienta es desarrollada por IBM para la gestión de requisitos y casos de uso de un proyecto, tiene una integración avanzada con Microsoft Word para familiarizar actividades como la especificación de requisitos, documentos Word para facilitar la sincronización de requisitos, integración y análisis de la organización.

Ventajas y desventajas:

- Proporciona información detallada de la trazabilidad de requisitos, que muestra vistas de las relaciones entre padres e hijos.
- Muestra las necesidades que se pueden afectar mediante algún cambio.
- Es poderosa y fácil herramienta para la gestión de requisitos.
- Es para plataforma Windows.

- Está integrado con otras suites de Rational.
- Permite crear vistas personalizadas de los requisitos almacenados en la aplicación y vistas de trazabilidad entre los requisitos.
- Permite configurar los tipos de requisitos que almacena el sistema.
- Almacena las actualizaciones o creaciones de requisitos.
- Hay que pagar licencia.
- Tiene plantillas para documentos y proyectos.
- Organización de los requisitos por paquetes y por vistas.
- Sincronización en Word y base de datos.
- Identificación de inconsistencia de forma gráfica, matriz de trazabilidad, matriz de trazabilidad textual, árbol de trazabilidad global.
- Modelo UML e integración de los requisitos.
- Comunicación de los cambios y su impacto. (Fernández, 2008)

GatherSpace:

Es una herramienta con entorno web para la gestión de requisitos utilizados para los trabajos en equipo distribuido.

Ventajas y desventajas:

- Creación de una jerarquía de requisitos: permite crear paquetes funcionales para luego relacionarlos con componentes de más alto nivel para luego permitir asociar casos de uso más detallados y requisitos de software a dichos componentes
- Consta de módulos para gestionar varios proyectos al mismo tiempo controlando el acceso de los usuarios a cada uno de los proyectos y poder exportar la información contenida a uno por uno de los proyectos.
- Módulo de generación de reportes (visión y requisitos, modelo de casos de uso, detalles de características y requisitos asociados, casos de uso, detalles de requisitos, trazabilidad, cambios y errores).

- Visualiza la documentación generada a partir de los requisitos en tres formatos diferentes (HTML, PDF, Microsoft Word).
- Módulo de consulta a través de la aplicación de filtros.
- La aplicación no es un paquete de instalación por lo que corre en servidores fuera del lugar de trabajo.
- Es privativa y se debe comprar tanta cuenta como usuarios se prevea tenga la aplicación. (Fernández, 2008)

IrqA:

Es una herramienta diseñada para la gestión de requisitos y es una de la más completa en el mercado.

Ventajas y desventajas:

- Capacidad de recuperar requisitos de documentos Word.
- Las descripciones de los requisitos pueden referenciar a documentos externos (tablas, gráficos, hojas de cálculo).
- Se puede hacer relaciones entre los requisitos.
- Permite exportar e importar a UML.
- Está integrado con Rational Rose.
- Posibilidad de ampliar la funcionalidad de la herramienta y acceder a la información de otras herramientas (API basadas en componentes COM y Java)
- Posibilidad de clasificación de requisitos, clasificación jerárquica, basada en el dominio o gestión de atributos, basada en atributos definidos por el usuario, basada en la relación con otros componentes de la especificación.
- Visualización gráfica, matriz de trazabilidad, vista de elementos relacionados.
- Proporciona algunas herramientas que verifican que todos los requisitos son satisfechos y donde.
- Ofrece un soporte específico para las pruebas de aceptación y validación.
- Reutilización de requisitos. (Fernández, 2008)

Raquest:

Es una herramienta para el análisis y gestión de requisitos, permitiendo tanto el manejo de requisitos del sistema como de aplicaciones.

Ventajas y desventajas:

- Admite manejar y definir requisitos con un gran número de propiedades (descripción, versión, ID, dificultad, prioridad, estabilidad y riesgos).
- No permite dar un rápido vistazo de los requisitos definidos.
- Posibilita la creación de paquetes para agrupar requisitos.
- Genera documentos de parte o de todo el proyecto en distintos formatos como son HTML, Word, Excel.
- Distingue los requisitos que son aprobados.
- No permite imprimir listas de los requisitos definidos en cualquier momento.
- No permite definir un glosario de términos.
- Es disponible solo para Windows.(Fernández, 2008)

Con el estudio realizado de las diferentes herramientas más utilizadas en el mundo se puede decir que no todas solventan las especificidades requeridas de la Universidad de las Ciencias Informáticas (UCI), ya muchas son privativas, las que son libres y de código abierto (Open Source) no cubren todas las fases de la ingeniería de requisitos. OSRMT (Open Source Requirements Management Tool) es una herramienta muy poderosa ya que es libre, de código abierto y multiplataforma pero no cumple con las matrices de trazabilidad que se utilizan en la fase de administración, carece de poco soporte y poca documentación. Es de ahí que se propone el desarrollo de una nueva herramienta que cumpla con los requisitos solicitados, seguidamente se proponen una serie de características para su desarrollo.

Posibles características para el sistema propuesto

- Código abierto, gratuita y multiplataforma.
- Que defina glosario de términos.

- Que sea una aplicación web.
- Generar documentos de parte o de todo el proyecto en distintos tipos de formatos.
- Organizar los requisitos por paquetes y vistas.
- Visualización gráfica, matriz de trazabilidad.
- Reutilización de requisitos.
- Posibilidad de ampliar las funcionalidades de la herramienta.
- Capacidad de recuperar requisitos de documentos Word.
- Visualizar la documentación generada en diferentes tipos de formatos.
- Modelo UML e integración con los requisitos.
- Almacenar las actualizaciones y creación de los requisitos.
- Que sea fácil y poderosa para la gestión de requisitos.
- Que muestre las necesidades que pueden surgir mediante algún cambio.
- Historia en los cambios de requisitos de manera automática.
- Conste de una interfaz intuitiva.
- Posibilidad de enlace con otra herramienta.
- Permita definir el nivel de accesibilidad de los usuarios al sistema.
- Llevar un control de quién y cuándo ha realizado un cambio sobre la documentación.
- Una interfaz de usuario rápida.
- Llevar el control de versiones de la información registrada en el sistema.
- Identificar inconsistencias de forma gráfica.

1.6.Patrones de casos de uso.

Actualmente los patrones de casos de uso son muy importantes en el desarrollo de un software. La utilización de patrones se considera como una buena práctica durante el análisis por tal motivo a continuación se especifican algunos patrones importantes que se deben utilizar para la realización del análisis.

A través de los años los patrones de casos de uso han ido evolucionando por la utilización que tienen en los proyectos. Estas técnicas han demostrado ser la solución preferida por todos los desarrolladores de software.

A continuación se presentan una descripción detallada de una serie de patrones de casos de usos más usados:

CRUD: completo

Es un patrón de estructura como bien su nombre lo indica (crear, leer, actualizar y eliminar) este se utiliza donde se quiere realizar altas, bajas, cambios y consultas a alguna entidad del sistema y se identifica con un caso de uso llamado **Gestionar Información**. Este patrón debe ser usado cuando todas las operaciones forman a un mismo valor de negocio, siendo todas cortas y simples.

Reglas del Negocio:

Es un patrón de estructura que se basa en la extracción de información originada de las políticas, reglas y regulaciones del negocio de la descripción del flujo y describe la información como una colección de reglas del negocio referenciadas a partir de las descripciones de los casos de uso.

Múltiples actores:

Roles diferentes

Captura la concordancia entre actores manteniendo roles separados. Consiste en un caso de uso y por lo menos dos actores. Es utilizado cuando dos actores juegan diferentes roles en un caso de uso.

Roles comunes

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

Extensión o Inclusión Concreta: Inclusión

En este patrón, existe una relación de inclusión desde el caso de uso base al caso de uso incluido. El último puede ser instanciado por el mismo. El caso de uso base podría ser cualquiera, concreto o abstracto. El patrón es usado cuando un flujo podría ser incluido en el flujo de otro caso de uso y además puede ser realizado por el mismo.

Extensión o Inclusión Concreta: Extensión

El patrón Extensión o Inclusión Concreta: extensión consiste en dos casos de uso y una relación de extensión entre ellos. El caso de uso extendido es concreto; es decir, puede ser instanciado por el mismo, así como también se puede extender del caso de uso base.

El patrón es aplicable cuando un flujo puede extender el flujo de otro caso de uso, así como también puede ser realizado por el mismo.

Concordancia:

Extrae una subsecuencia de acciones que surgen en diferentes lugares del flujo del caso de uso y es expresado por separado (Reuso, Adición, Especialización). (ÖVERGAARD & Karin, 2004)

1.7.Métricas para el Análisis.

El principal objetivo del análisis es transformar los requisitos obtenidos en el sistema a implementar. El análisis se centra fundamentalmente en la modelación de los requisitos funcionales para su implementación en el sistema, mientras que el diseño asegura que se cumplan en la aplicación los requisitos no funcionales.

Para estimar el tamaño del software durante el análisis se aplican los puntos de función como una base para aplicar métricas en próximas fases del desarrollo de software. La métrica de flujo de información se utiliza como medida de facilidad de mantenimiento y prueba del software, esto ofrece una medida de la complejidad del diseño del modelo de datos y la complejidad ciclomática.

Puntos de función

Esta métrica es una de las más conocidas y eficientes basada en pesos. Para calcular esta métrica primero se debe completar la tabla (ver figura 3).Se determinan 5 características de dominio de información y se proporcionan las cuentas en la posición apropiada de la tabla.

		Factor de ponderación			
Parámetro de medición	**Cuenta**	**Simple**	**Medio**	**Complejo**	
Numero de entradas de usuario	☐ x	3	4	6	= ☐
Numero de salidas de usuario	☐ x	4	5	7	= ☐
Numero de peticiones de usuario	☐ x	3	4	6	= ☐
Numero de archivos	☐ x	7	10	15	= ☐
Numero de interfaces externas	☐ x	5	7	10	= ☐
Cuenta = Total	→				☐

Figura 3.Cálculo de métricas de punto de función.

Los valores de los dominios de información se definen de la forma siguiente:

- Número de entrada del usuario: número de entrada que el usuario le proporciona al software. Esto representa la habilidad de controlar y mantener el procesamiento de datos.
- Número de salida del usuario: número de salida que proporcionan información al usuario. Representa la salida de información de la aplicación al usuario.
- Número de archivos lógicos internos: número de colecciones lógicas de datos mantenidas desde la aplicación.
- Número de archivos de interfaz externa: número de colecciones lógicas de datos mantenidas fuera de la aplicación, pero referenciadas por ella.
- Número de consultas del usuario: número de elementos que el usuario puede solicitar a la aplicación.

Las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada es simple, media o compleja.

Para calcular los puntos de función (PF) se utilizan las relaciones siguientes:

PF=cuenta-total x [0,65+0.01x 6 (F_i)] en donde cuenta-total es la suma de todas las entradas PF obtenidas de la figura 3.

F_i ($_i$=1 a 14) son <<valores de ajuste de la complejidad >> basados en las respuestas a las cuestiones señaladas de la siguiente tabla (ver figura 4). Evaluar cada factor en escala 0 a 5. (Pressman, 2005)

0	1	2	3	4	5
Sin influencia	Incidental	Moderado	Medio	Significativo	Esencial

Fi:

1. ¿Requiere el sistema copias de seguridad y recuperación fiables?
2. ¿Se requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es critico el rendimiento?
5. ¿Será ejecutado el sistema en un entorno operativo existente y frecuentemente utilizado?
6. ¿Requiere el sistema entrada de datos interactivo?
7. ¿Requiere la entrada de datos interactivo que las transiciones de entrada se lleven a cabo sobre múltiples o variadas operaciones?
8. ¿Se actualizan los archivos maestros en forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidos en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Figura 4. Tabla y cuestiones para medir puntos de función.

Métricas de la calidad de la especificación

Para valorar la calidad del modelo de análisis y la correspondiente especificación de requisitos se proponen una lista de características: especificidad (ausencia de ambigüedad), compleción, corrección,

comprensión, capacidad de verificación, consistencia interna y externa, capacidad de logro, concisión, trazabilidad, capacidad de modificación exactitud y capacidad de reutilización. Aunque muchas de estas características parecen ser de naturaleza cuantitativa se sugiere que todas puedan representarse usando una o más métricas. Por ejemplo, asumiendo que hay $\mathbf{n_r}$ requisitos en una especificación tal como $n_r=n_f+n_{nf}$.

Donde n_f es el número de requisitos funcionales y n_{nf} es el número de requisitos no funcionales.

Para determinar la especificidad de los requisitos se sugiere una métrica basada en consistencia de la interpretación de los revisores para cada requisito: $Q_1 = n_{ui}/n_r$

Donde n_{ui}es el número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas. Cuanto más cerca este de 1 el valor de Q menor será la ambigüedad de la especificación.

La compleción de los requisitos funcionales puede determinarse calculando la relación:

$Q_2=u_n/ (n_i \times n_s)$

Donde u_n es el número de requisitos únicos de función, n_i es el número de entradas (estímulos) definidos o implicados por la especificación y n_s es el número de estados especificados. La relación Q_2 mide el porcentaje de funciones necesarias que se han especificado para un sistema. Sin embargo, no trata los requisitos no funcionales. Para incorporarlos a una métrica global completa debemos considerar el grado de validación de los requisitos.

$Q_3 = n_c/ (n_c + n_{nv})$ donde $_c$es el número de requisitos que se han validado como correctos y n_{nv} el número de requisitos que no se han validado todavía.

La especificación de requisitos de software (SRS), es una corrección, si y sólo si, todos los requisitos necesarios representan algo del sistema que se construirá. Cada requisito en el SRS contribuye a la satisfacción de alguna necesidad.

Ya que la corrección se aplica a un individuo y una obligación de toda SRS, una manera conveniente de medir la exactitud de un SRS podría consistir en medir el porcentaje de forma individual de los requisitos de corrección, es decir, nc /nc + nl donde nl y nc son la cantidad de requisitos correctos e incorrectos

respectivamente, y nr = nc + nl. El valor del rango está entre 0 (totalmente incorrecto) a 1 (totalmente correcto).

Irónicamente, si hubiéramos podido saber que los requisitos no son correctos, se podría eliminar, ¡por lo que es 100% correcto! aplicando así la fórmula anterior, siempre dará lugar a una puntuación de 1. Uno más práctico, pero menos teóricamente satisfactoria, es medir el porcentaje de los requisitos en el SRS que hayan sido validados.

1.8.Conclusiones.

En este capítulo se realizó un estudio sobre varias metodologías de desarrollo de software. Se destellaron los distintos tipos de patrones de casos de uso más usados. Se realizó un análisis de diferentes herramientas CASE, resaltando en las posibilidades que brindan, se estudió sobre el lenguaje de modelado.

Luego de tener todos estos datos se arribaron a las siguientes conclusiones:

- Utilizar la metodología de desarrollo RUP. Por ser una guía eficiente para realizar el análisis.
- En la ingeniería de requisitos se aplicarán las 5 etapas (elicitación, análisis, especificación, validación y administración), aplicando cada actividad que esta propone.
- Aplicar patrones de casos de uso ya que brindan una gran ayuda a la hora de especificar, identificar y crear diagramas de casos de uso para la herramienta que nos brinda una buena gestión de requisitos.
- Utilizar el lenguaje UML de modelado ya que permite todos los modelos necesarios para el análisis de la herramienta.
- Utilizar como herramienta CASE de desarrollo a Visual Paradigm versión 6.4 por permitir la modelación del negocio, sistema y prototipo de interfaz de usuario del análisis de las herramientas.

Capítulo 2. Solución propuesta. Análisis del sistema

2.1.Introducción.

El presente capítulo muestra la solución que se propone para el análisis de la herramienta para la gestión de requisitos. Comenzando a partir de las técnicas aplicadas de la ingeniería de requisitos para obtener las funcionalidades y reglas del sistema a partir de los procesos de negocio. Seguidamente se especifican los artefactos utilizados para la elaboración del modelo del sistema. Por último, se analizan los casos de uso, sus especificaciones y requisitos para modelar los artefactos necesarios del negocio y sistema, logrando así poder desarrollar un adecuado modelo de diseño.

Modelo del negocio

El modelado del negocio permite definir los procesos, roles y responsabilidades de la organización en los modelos de casos de uso del negocio. De aquí que este proceso esté relacionado con los de obtención de requisitos y análisis. Es considerado una técnica importante para la especificación de requisitos con lo cual se refuerza la idea de que sea el propio negocio lo que determine los requisitos. Durante el modelado se obtienen artefactos importantes como los diagramas de actividad y los modelos de casos de uso del negocio. Todo esto es importante para el modelamiento del sistema.

Modelo del sistema

El modelo del sistema tiene como principal objetivo definir cuáles son los requisitos funcionales y no funcionales que tendrá el mismo. Luego de tener definido lo que es el alcance del sistema, se pasa a encontrar los actores, seguidamente los casos de uso y la especificación de cada uno de ellos. Todo esto conjuntamente con el modelo de casos de uso del sistema ayuda posteriormente a obtener un modelado del diseño.

2.2.Modelo de negocio.

Actores del negocio

- Usuario
- Cliente
- Desarrollador

Trabajadores del negocio

- Analista
- Revisor
- Usuario

Diagrama de casos de uso del negocio

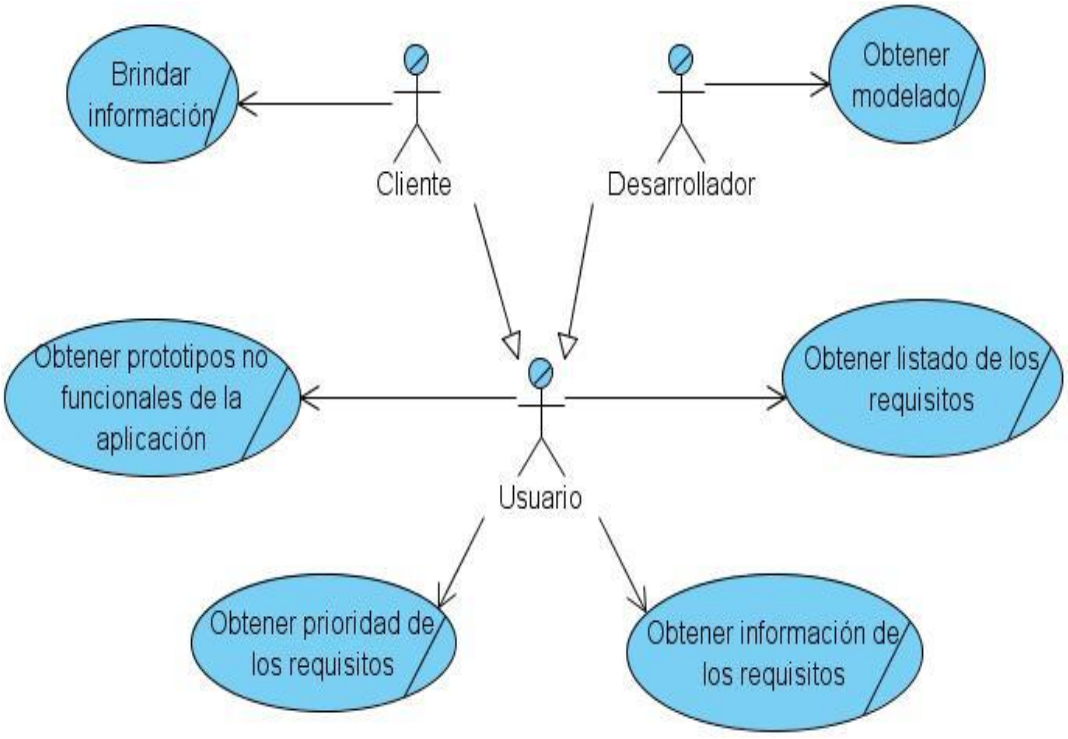


Figura 5. Diagrama de casos de uso del negocio.

Especificación de los casos de uso

Tabla 2. Descripción textual del caso de uso del negocio Brindar información.

Nombre del caso de uso del negocio:	Brindar información
Actores del negocio:	Cliente
Propósito:	Obtener toda la información necesaria para que los analistas puedan encontrar los requisitos del sistema.
Resumen: El procedimiento de brindar información comienza cuando la persona brinda alguna información, el analista obtiene la información y encuentra los posibles requisitos del sistema.	
Casos de uso asociados:	-
Flujo de trabajo	
Acción del Actor	**Respuesta del negocio**
1. El caso de uso inicia cuando el actor cliente brinda la información.	2. El analista recibe la información, la analiza y selecciona los requisitos del sistema.
3. Analiza los requisitos y los acepta o los rechaza, en caso de ser rechazado notifica y propone nuevos requisitos.	4. El caso de uso termina.

Las especificaciones de los casos de uso restantes se pueden encontrar en el anexo A.

Diagrama de actividades de los casos de uso

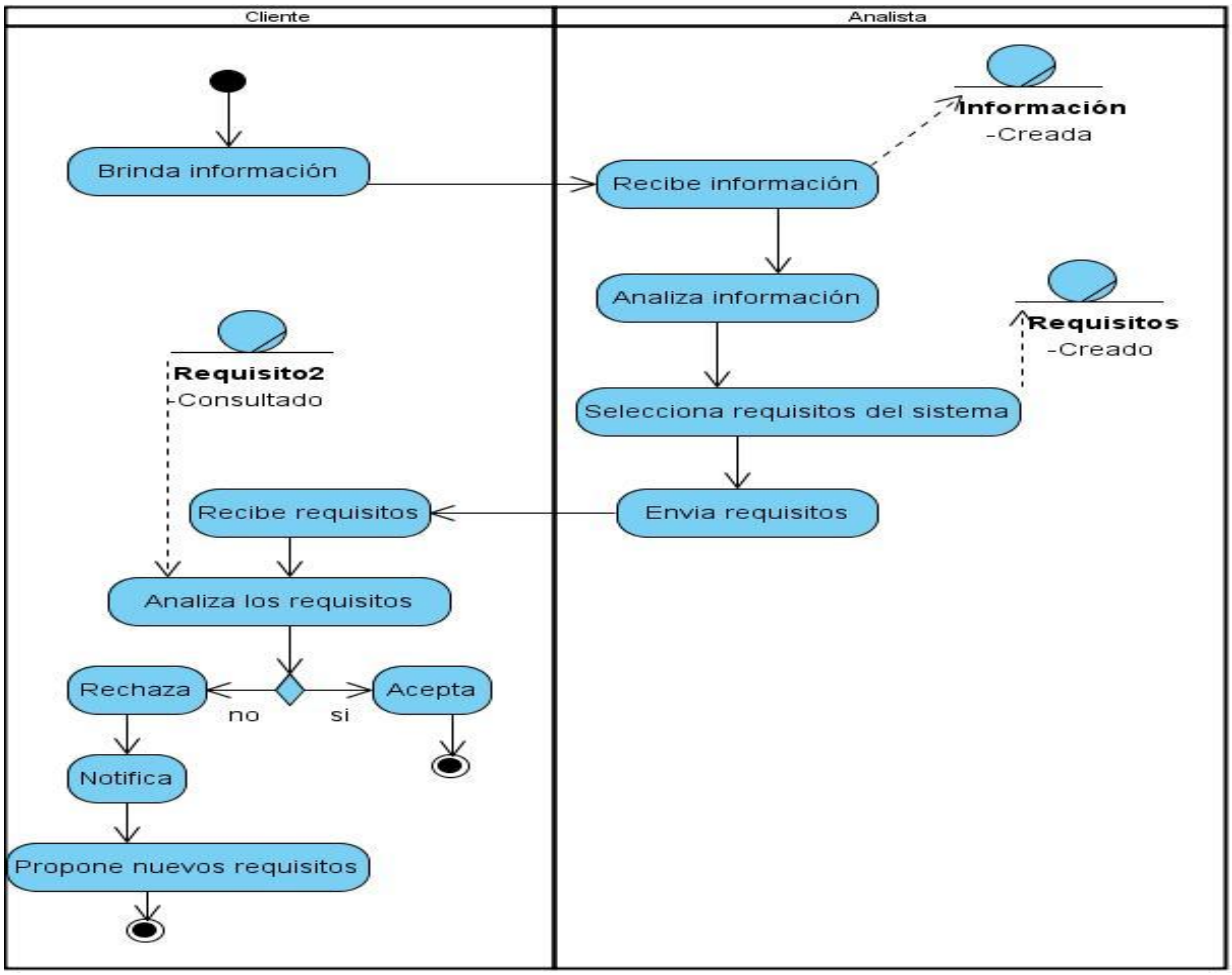


Figure 6. Diagrama de actividad del caso de uso Brindar información.

Los diagramas de actividad restantes se pueden encontrar en el anexo F.

2.3.Requisitos de software.

Con la aplicación de técnicas para la elicitación de requisitos y la interacción con los clientes, se obtuvieron los requisitos que debe cumplir el sistema así como las restricciones necesarias. Se identificaron las necesidades de los usuarios y clientes con el fin de obtener una herramienta que satisfaga estas necesidades.

Los requisitos obtenidos son analizados de tal manera que se llegue a un entendimiento entre los clientes y desarrolladores de cómo será el producto de manera funcional. Estos son identificados en dos categorías, funcionales y no funcionales. Llevando un control de la calidad de las especificaciones de los mismos.

2.3.1. Funcionales.

Los requisitos funcionales no son más que condiciones o capacidades que el sistema debe cumplir, obtenidas mediante las técnicas aplicadas de la elicitación.

A continuación se presentan los requisitos funcionales de la herramienta para la gestión de requisitos que se propone. Los mismos serán identificados con las siglas RF_N. la n significa el Número del requisito (Ej. RF_ 1) y clasificados según su prioridad en Alta (Esencial), Media (Deseado) o Baja (Opcional).

RF_9 Crear requisito.

El sistema permitirá crear un requisito especificando los siguientes datos:

- Nombre.
- Descripción.
- Fecha.
- Prioridad.
- Identificador.

Prioridad: alta.

RF_34 Buscar requisitos.

El sistema permitirá buscar requisitos a partir de los siguientes criterios de búsqueda:

- Palabra clave.
- Fecha de creación.
- Analista responsable.

- Cliente que brindó la información.

Prioridad: alta.

RF_35 Proponer nuevos requisitos.

El sistema permitirá hacer nuevas propuestas de requisitos a partir de listados mostrados de los mismos, mediante los siguientes datos.

- Palabra clave.
- Fecha de creación.
- Analista responsable.
- Cliente que brindó la información.

Prioridad: alta.

RF_45 Eliminar cambios en los requisitos.

El sistema permitirá eliminar los cambios realizados en un requisito que sea seleccionado.

Prioridad: alta.

RF_49 Crear control de versiones.

El sistema deberá permitir realizar un control de las versiones de los cambios aprobados en los requisitos individuales o grupos de ellos y definir la línea base, a partir de los siguientes datos.

- Nombre.
- Descripción.
- Fecha.
- Prioridad.
- Línea base.

Prioridad: alta.

RF_50 Eliminar control de versiones.

El sistema deberá permitir eliminar el control de las versiones de los cambios aprobados en los requisitos individuales o grupos de ellos.

Prioridad: media.

RF_52 Listar control de versiones.

El sistema permitirá listar el control de las versiones de los cambios aprobados en los requisitos individuales o grupos de ellos, de los mismos se especificara.

- Nombre.
- Descripción.
- Fecha.
- Prioridad.

Prioridad: alta.

Las especificaciones de los requisitos funcionales restantes se pueden encontrar en el anexo B.

2.3.2. No Funcionales.

Lo requisitos no funcionales son cualidades o propiedades que el sistema debe cumplir.

Usabilidad

RNF_1 Cumplir con las pautas de diseño de las interfaces.

RNF_2 Agrupar vínculos y botones por grupos funcionales.

Fiabilidad

RNF_8 Prever contingencias para eventos de caída del sistema.

Eficiencia

RNF_9 Responder en tiempos aceptables las peticiones que se realicen en el sistema.

Seguridad

RNF_10Permitir el intercambio de datos entre el cliente y el servidor por canales cifrados.

RNF_11 Almacenar de manera cifrada las claves de los usuarios en la base de datos.

Portabilidad

RNF_21 Permitir la configuración de cada uno de los nomencladores del sistema.

Reusabilidad

RNF_29 Garantizar que los formatos de los archivos de salida del sistema sean compatibles con los programas más comunes.

RNF_30 Definir un modelo tres capas para el sistema

Capacidad

RNF_31 Considerar características técnicas mínimas para la ejecución en clientes

La especificación de los requisitos no funcionales se puede encontrar en el anexo C.

2.4.Actores del sistema.

Al definir la frontera del análisis de la herramienta para la gestión de requisitos se encontraron los actores del sistema que representan cada trabajador del negocio que tiene actividades a automatizar. Además de los actores del negocio que interactúan de alguna forma con el sistema.

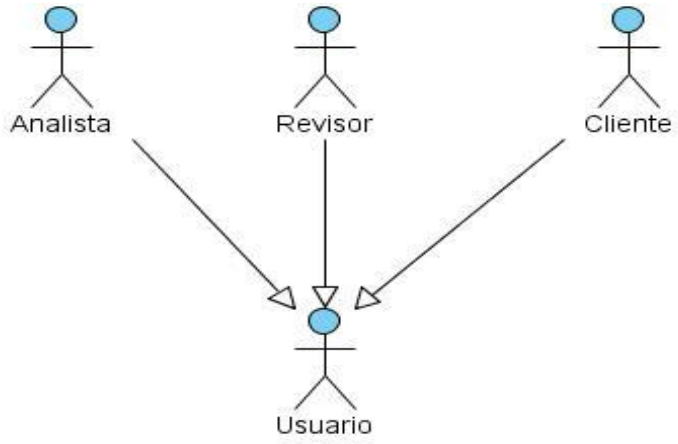


Figura 7. Actores del sistema

2.5.Diagrama de casos de uso del sistema.

Luego de identificar los requisitos y especificarlos se estructura el diagrama de casos de uso del sistema aplicándole los patrones de caso de uso especificados en el epígrafe 1.4. Estos se organizaron por paquetes de la siguiente forma.

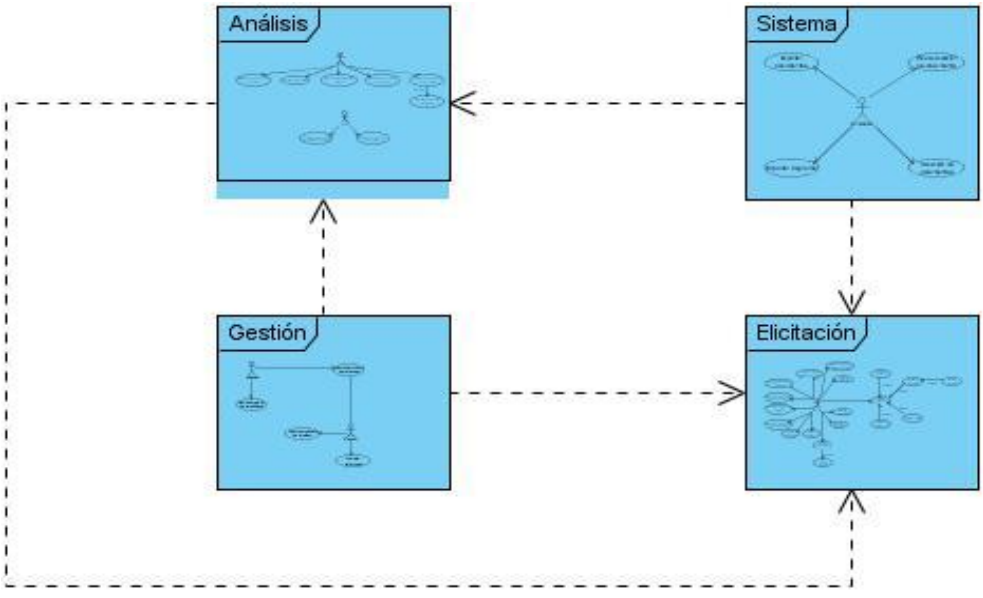


Figure 8. Paquetes del software Herramienta para la Gestión de Requisitos.

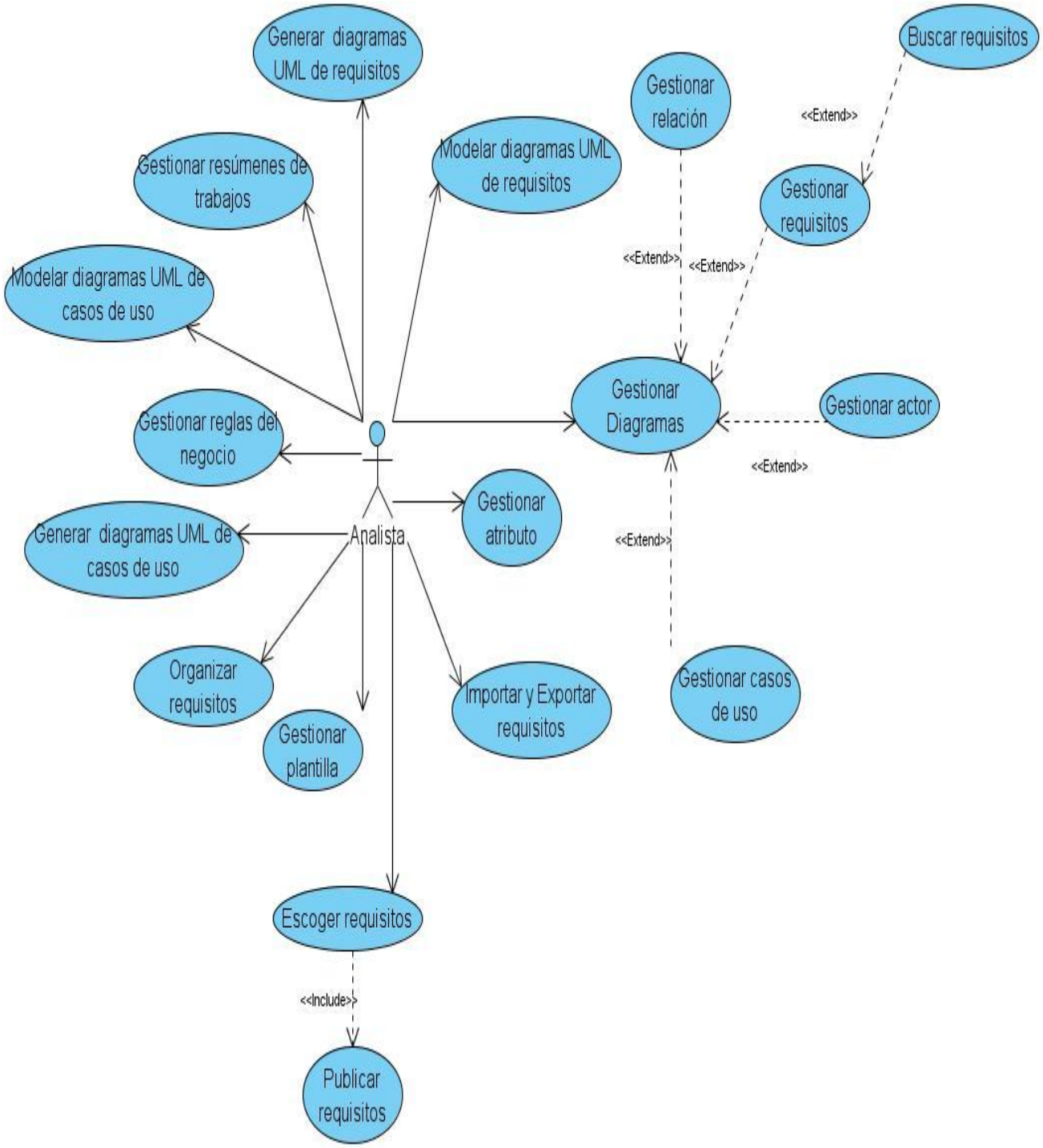


Figure 9. Diagrama de casos de uso del paquete Elicitación

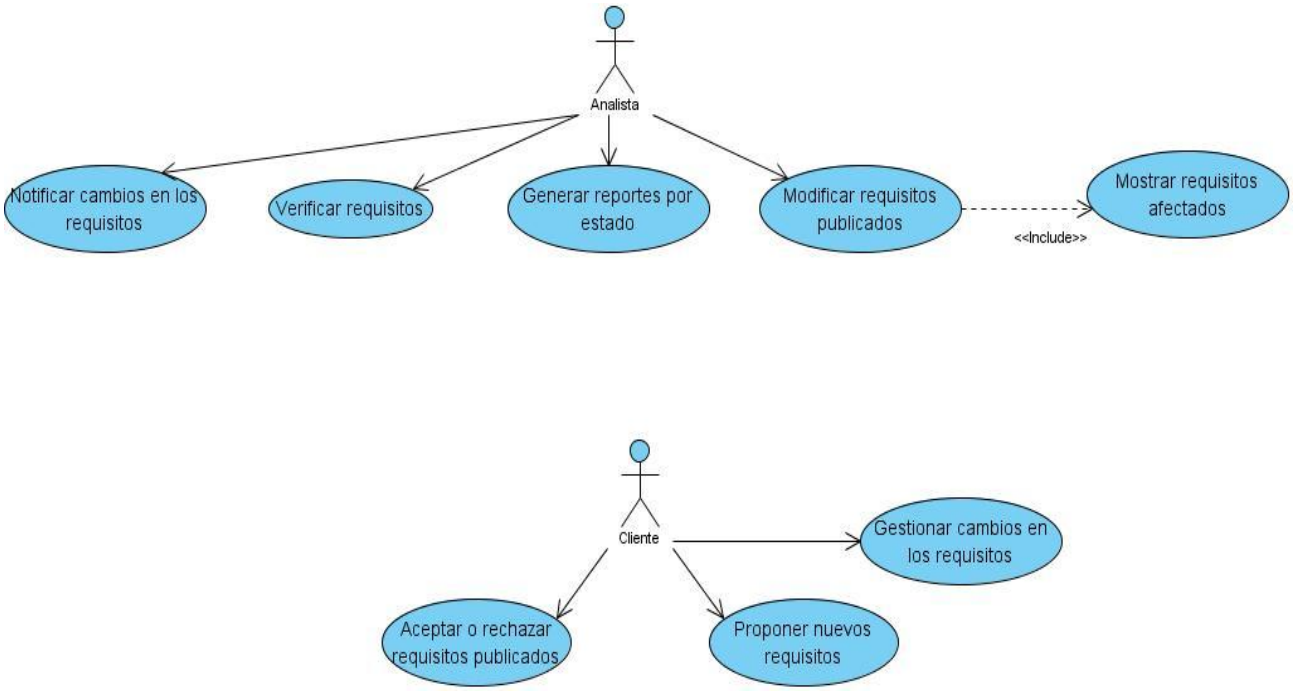


Figure 10. Diagrama de casos de uso del paquete Análisis

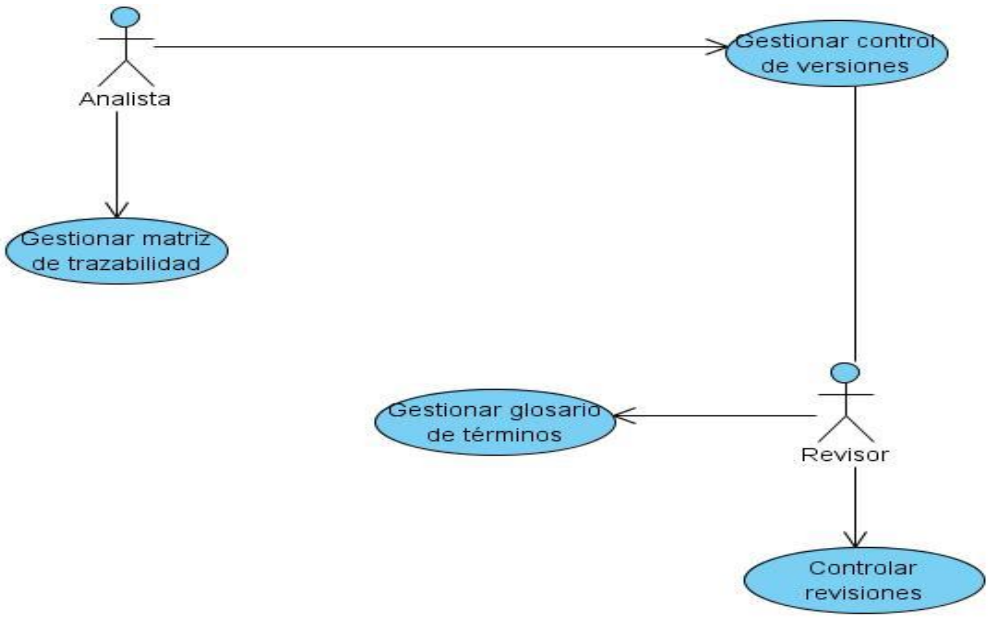


Figure 11. Diagrama de casos de uso del paquete Gestión.

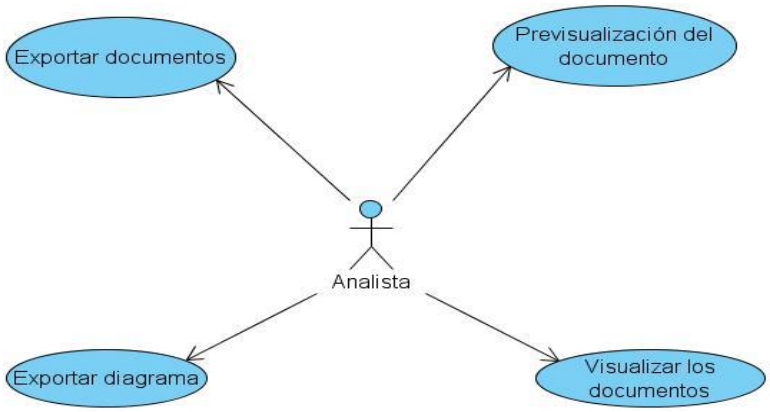


Figure 12. Diagrama de casos de uso del paquete Sistema

2.6.Administración de requisitos.

Para seguir y llevar un control de los cambios en los requisitos en esta actividad se lleva a cabo la técnica de trazabilidad de requisitos. Esta técnica trata de hacer una relación entre los requisitos del sistema con sus casos de uso.

La figura siguiente muestra una parte de la matriz de trazabilidad de requisitos, construida a partir de los casos de uso del sistema y los requisitos funcionales especificados anteriormente.

Tabla 3. Porción de la matriz de trazabilidad de requisitos.

	Casos de Uso											
REQ.	CU18	CU19	CU20	CU21	CU22	CU23	CU24	CU25	CU26	CU27	CU28	CU29
RF_1								X				
RF_35											X	
RF_36											X	
RF_37											X	
RF_38						X						
RF_50					X							
RF_55												X
RF_56										X		
RF_57									X			
RF_61												
RF_62				X								
RF_63												

RF_65	x											
RF_66		x										
RF_67			x									
RF_68												
RF_69												
RF_80							x					
RF_86												

La matriz de trazabilidad completa se puede encontrar en el anexo D.

2.7.Descripción de casos de uso del sistema.

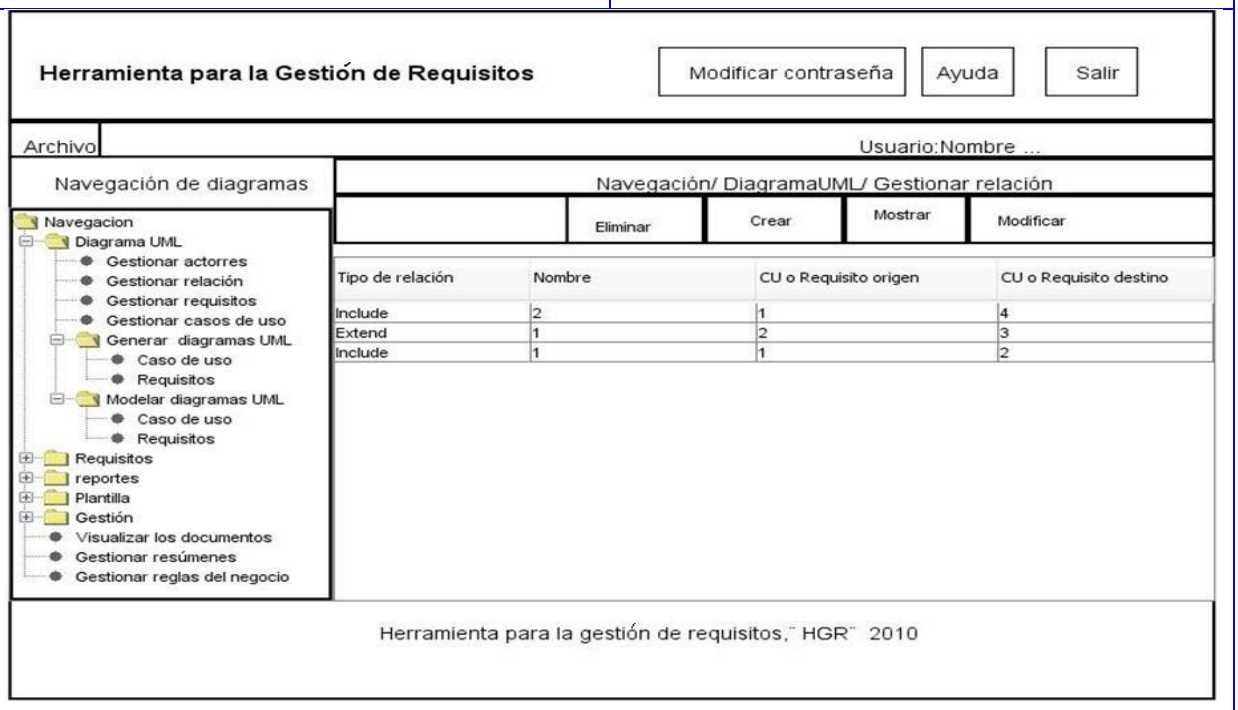
Una vez identificado los actores y casos de uso del sistema, se pasa a especificar los pasos necesarios para cumplir con dichos casos de uso. Indicando en cada uno los eventos que van ocurriendo durante el mismo, seguidamente se presenta la especificación de un caso de uso del sistema.

Tabla 4. Resumen de la descripción textual del caso de uso del sistema Gestionar relación.

Caso de Uso:	Gestionar relación
Actores:	Analista
Resumen:	El caso de uso se inicia cuando el Analista necesita gestionar una relación. Consiste en que el Analista selecciona la opción de crear una nueva relación y llena los campos requeridos para la creación. También tiene la posibilidad de seleccionar una relación ya sea para modificarla o eliminarla. El caso de uso termina con la creación, modificación o eliminación de una relación.
Precondiciones:	• El sistema debe estar instalado y ejecutándose correctamente. • El usuario debe estar autenticado con los permisos necesarios.
Referencias	RF_3, RF_ 4, RF_ 5, RF_6, RF_7
Prioridad	Crítico
Complejidad	Complejo

<table>
<tr><td>Nivel del caso de uso</td><td colspan="2">Usuario</td></tr>
<tr><td colspan="3">Flujo Normal de Eventos</td></tr>
<tr><td colspan="2">Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2">1. El caso de uso inicia cuando el actor Analista selecciona la opción “Diagrama UML” y dentro de esta la opción “Gestionar relación”.</td><td>2. El sistema muestra una interfaz donde se lista el conjunto de Relaciones existentes. De las mismas se muestra:
Tipo de relación.
Nombre.
CU o Requisito origen.
CU o Requisito destino</td></tr>
<tr><td colspan="2">3. El actor Analista selecciona una de las siguientes opciones:
Crear (Ver sección “Crear nueva relación”)
Modificar (una vez que haya seleccionado el problema que desea modificar, ver sección “Modificar relación”)
Eliminar (una vez que haya seleccionado el problema que desea eliminar, ver sección “Eliminar relación”)
Mostrar (una vez que haya seleccionado el problema del que</td><td></td></tr>
</table>

desea ver sus detalles, ver sección **"Mostrar relación"**)	



Las especificaciones completas de los casos de uso se encuentran en el anexo E.

2.8.Conclusiones.

- Usando RUP y utilizando sus principales características se generaron los artefactos necesarios durante el análisis.
- Se identificaron las funcionalidades y características, reflejadas en el modelo del sistema donde se especifican los actores y las acciones que estos realizan.
- Se aplicaron patrones de casos de uso para facilitar la estructura del diagrama de casos de uso del sistema.
- Se aplicaron la matriz de trazabilidad para el control y seguimiento de los requisitos y sus cambios.

Capítulo 3. Análisis de los resultados

3.1.Introducción.

En este capítulo se aplican métricas y técnicas para medir la calidad de los requisitos identificados. Para ello se llevan a cabo los procesos de verificación y validación de requisitos así como la validación de los casos de uso.

3.2.Verificación de requisitos.

Verificación: el conjunto de actividades orientadas a alcanzar la calidad interna exigible a las especificaciones de requisitos conforme a las normas establecidas por la organización previamente.

Para la verificación de los requisitos se llevaron a cabo cuatros actividades fundamentales: obtención de las métricas para los requisitos, revisión de los requisitos a partir de las métricas obtenidas, análisis del progreso de los requisitos y lista de chequeo aplicada a los requisitos para verificar su calidad, esta última propuesta por el grupo de Calidad de la Universidad de las Ciencias Informáticas (UCI). (Award Winning UML Tool. Visual Paradigm for UML User's Guide, 2007).

Las métricas identificadas se aplicaron para verificar la existencia de un grupo de propiedades que debe cumplir la especificación de requisitos para garantizar su calidad, estas propiedades fueron propuesta por Davis en el modelo de calidad que él define (A. Davis, 1993). Para la verificación de la especificación de casos de uso se aplicaron preguntas para comprobar la presencia de un conjunto de propiedades a partir de las propuestas de Davis (A. Davis, 1993), Cockburn (Cockburn, 2001), y las listas de chequeo propuesta por calidad UCI.

La primera de las propuestas se consideró por recoger varias de las propiedades que se utilizaron para la especificación de requisitos y que son completamente válidas para la especificación de los casos de uso. La segunda y tercera se utilizaron por ser específicas para la especificación de casos de uso; como es el caso de las listas de chequeo de calidad UCI, que verifican el uso correcto de las relaciones entre casos de uso y la propuesta de Cockburn por constituir una de las guías más exhaustivas en el buen uso de las técnicas de casos de uso.

3.3.Modelos de calidad para requisitos.

Según ([ISO/IEC1991]), un modelo de calidad es la descripción, en términos de un conjunto estructurado de características y sub-características, de los atributos que debe tener un producto software y que contribuyen a que dicho producto sea bueno dentro de los de su clase.

A la hora de presentar un modelo de calidad para ingeniería de requisitos, las propuestas que se han estudiado optan por una de las siguientes opciones: aportar una lista de características de calidad de los requisitos, o en contraposición, proporcionar una taxonomía de defectos, entendiendo que un defecto en los requisitos es la ausencia de alguna de las características de calidad.

Para la verificación de los requisitos se utilizó el tipo de modelo de calidad que propone una lista de características. Uno de los trabajos más citados en lo que respecta a las clasificaciones de las características de calidad en los requisitos es la propuesta de Davis. Su propuesta presenta un conjunto de veinticuatro propiedades deseables de los requisitos, además de ciertas métricas para verificar hasta qué punto los requisitos de una especificación cumplen esas propiedades. En el presente trabajo se realizó una selección de las que se consideraron de mayor importancia y se aplicaron las métricas correspondientes para verificar la presencia de dichas propiedades.

Propiedades:

No redundancia: cada requisito aparece una sola vez.

Consistencia interna: no existe ningún subconjunto de requisitos que incluyan conflictos.

Consistencia externa: ningún requisito entra en conflicto con otra documentación del proyecto.

No ambigüedad: tiene una única interpretación posible.

Reutilizabilidad: las frases, párrafos y secciones pueden ser fácilmente adoptadas o adaptadas para futuras especificaciones.

Corrección: todo requisito contribuye a la satisfacción de una necesidad y todas las necesidades se hallan recogidas en la especificación

3.4.Métricas para la verificación de la especificación de requisitos.

En la siguiente tabla se muestran las métricas auxiliares obtenidas y aplicadas como resultado de la primera y segunda actividad (obtención de las métricas y revisión de los requisitos) de la metodología aplicada.

Tabla 5. Métricas auxiliares aplicadas a la especificación de requisitos

Métrica	Descripción	Valor
RT	Total de Requisitos	107
RC	Cantidad de requisitos cambiados (suma de los requisitos insertados, modificados y eliminados)	11
NUI	Número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.	100
RRT	Cantidad de requisitos reutilizados	26
RCI	Cantidad de requisitos en conflictos internamente en la especificación.	0
RCE	Cantidad de requisitos en conflictos con otra documentación del sistema.	0
CRR	Cantidad de requisitos repetidos	0
NC	Cantidad de requisitos correctos	107
NI	Cantidad de requisitos incorrectos	0
NR	Cantidad de requisitos bien especificados	100

A continuación se muestran las métricas principales aplicadas a la especificación de requisitos con las respectivas propiedades que verifican según la propuesta de Davis.

Tabla 6.Métricas principales aplicadas a la especificación de requisitos

No	Métrica	Formula	Propiedad	Valor
1	Corrección	NR = NC/NC+NI* 100	Corrección	100%

2	Reutilización	RR = RRT / RT * 100	Reutilizabilidad	30 %
3	Requisitos en conflictos	PRC= (RCI+RCE)/RT* 100	Consistencia interna y externa	0 %
4	Por ciento de redundancia	RR = CRR / RT * 100	No redundancia	0 %
5	Especificidad	Q = NUI / RT * 100	No ambigüedad	97,6%

Si se observa detenidamente los resultados de las tablas 4 y 5, se puede apreciar que el grado de reutilización de la especificación de requisitos de herramientas que tratan el tema de gestión de requisitos es medio si se tiene en cuenta que la herramienta a desarrollar es de nueva creación.

Un alto por ciento de reutilizabilidad se logra en gran medida si la especificación en herramientas orientadas a la gestión de requisitos se realiza con vista a una futura reutilización, utilizando para ello patrones de reutilización. Como se evidencia en este caso, no se dio la situación propicia para lograr altos niveles de reutilización, que es tan importante en esta etapa, de manera que se logre una estandarización cada vez más sólida de la especificación de requisitos.

En las tablas 4 y 5, es importante destacar que cuanto más cerca de 1 este el valor de la especificidad menos ambigüedad tendrá la especificación, lo cual ha sido planteado por Davis en su modelo de calidad, por lo que se puede decir en este caso que la especificación de los requisitos, carecen de un grado alto de ambigüedad.

Según Davis cuando el rango de la corrección está más cerca de cero es totalmente incorrecto los requisitos y entre más se acerca a uno es totalmente correcto. Por lo que se puede decir que los requisitos están a un grado de porcentaje correcto.

A continuación se muestra la lista de chequeo aplicada a los requisitos la misma se le aplica a las especificaciones de cada uno de los requisitos.

Tabla 7.**Lista de comprobación aplicada a la especificación de requisitos**

Evaluación	
Claridad	
¿Hay requisitos que tienen más de una interpretación?	no
¿Hay un glosario en el cual los requisitos significativos y específicos están en términos definidos y claros?	sí
¿Se entienden los requisitos para ponerlos en ejecución por un grupo independiente?	sí
Completo	
¿El requisito tiene un contenido específico?	sí
¿Se definen todos los términos en cada uno de los requisitos determinados?	sí
¿Tiene el artefacto un índice bien definido?	sí
¿Existen áreas donde está incompleta la información debido a que el desarrollo aún no ha comenzado a especificarse?	no
¿Algún requisito necesita una especificación detallada?	sí
¿Algún requisito necesita ser menos especificado?	sí
¿Todos los requisitos se describen ellos mismos?	no
¿Están incluidos todos los requisitos relacionados con la funcionalidad?	sí
¿Hay requisitos que produzcan inquietud?	sí
¿Están incluidos todos los requisitos relacionados con sus cualidades?	sí
¿Están incluidos todos los requisitos relacionados con las interfaces externas?	sí
¿Están incluidos todos los requisitos relacionados con el software?	sí
¿Están incluidos todos los requisitos relacionados con la seguridad?	no

Los resultados obtenidos al aplicar la lista de chequeo que se muestra en la tabla anterior evidencian que la especificación de los requisitos fue satisfactoria. El resto de la lista de comprobación aplicada se puede consultar en el anexo I.

3.5.Verificación de la especificación de casos de uso.

A continuación se muestra la lista de comprobación aplicada que verifica la calidad de la especificación de los casos de uso propuesta por el grupo de calidad de la Universidad de las Ciencias Informáticas. Las preguntas se han agrupado bajo una categoría relativa a una característica de calidad. En caso de alguna respuesta negativa, implica la presencia de un defecto en dicha especificación. En el proceso de revisión, la lista de comprobación se le aplica a las especificaciones de cada uno de los casos de uso.

Tabla 8. Lista de comprobación aplicada a la especificación de casos de uso

Evaluación	
¿Está debidamente identificada la plantilla?	sí
¿La plantilla cuenta con un Índice General, con una jerarquía de tópicos bien definida?	sí
¿La plantilla cuenta con una sección de Introducción?	sí
¿Está definido el Propósito de la modelación dentro de la introducción?	sí
¿Está definido el Alcance de la modelación dentro de la introducción?	sí
¿La plantilla cuenta con una sección donde se identifiquen y se describan los Actores del Sistema?	sí
¿La plantilla cuenta con una sección que recoja el diagrama de Casos de Uso del Sistema?	sí
¿La plantilla cuenta con una sección de Especificación de Casos de Uso?	sí
¿El formato de letra utilizado para la plantilla en su totalidad es uniforme y solo diferencia tópicas y conceptos que verdaderamente necesiten ser diferenciados del resto del texto?	sí
¿Es la documentación o redacción formal lo suficiente clara para ser entendida por personas fuera del equipo de proyecto?	sí
¿Cada descripción textual está conformada por una terminología única?	sí
¿Los párrafos están separados por dos espacios y justificados?	sí
¿Cada apéndice independiente está separado uno del otro con un espacio y ordenado desde la izquierda?	sí
¿Los apéndices están separados del texto anterior y posterior por tres y dos espacios respectivamente?	sí

¿La organización es de manera clara y lógica?	sí
¿La organización cumple con un estándar aceptado?	sí
¿La redundancia es mínima y sólo se debe a distintos niveles de abstracción o detalle?	sí
Actores del Sistema	
¿Hay correspondencia entre los actores del sistema y los trabajadores del negocio?	sí
¿Están descritos todos los actores de manera breve, concreta y formal?	sí

Los resultados obtenidos al aplicar la lista de chequeo que se muestra en la tabla anterior evidencian que la especificación de casos de uso fue satisfactoria.

El resto de la lista de comprobación aplicada se puede consultar en el anexo I.

3.6.Validación de requisitos.

Validación: es el conjunto de actividades encaminadas a llegar a un acuerdo entre todos los participantes en el que se ratifique que los requisitos elicitados, analizados y verificados representan realmente las necesidades de clientes y usuarios, por lo tanto, deberían llevar a la construcción de software útil.

Para la validación de los requisitos se llevaron a cabo tres actividades fundamentales: validación de los requisitos funcionales, validación de los requisitos no funcionales y cierre de la versión de los requisitos (Jacobson, 2000).

En la primera actividad, mediante walkthrough, asistidos con prototipos de interfaz de usuario no funcional, permitió una validación conjunta de forma sencilla guiada por los requisitos funcionales descritos como casos de usos.

De forma paralela a la primera actividad se realizó la segunda, con el objetivo de validar los requisitos no funcionales. Como resultado se obtuvo la aceptación y validación de los requisitos no funcionales, de forma similar a la actividad anterior.

Al igual que en las actividades de análisis, en las que los requisitos no funcionales no suelen contemplarse en las técnicas de modelado, en la validación tampoco parece existir una forma de integrarlos en un proceso de prototipado.

3.7.Conclusiones.

En este capítulo se ha hecho un análisis de los resultados obtenidos en el capítulo anterior arribando a las siguientes conclusiones:

- Se realizó la etapa de verificación y validación de los requisitos con los siguientes resultados:
  - Para la verificación se utilizaron métricas que dieron una medida del grado de calidad de la especificación de los requisitos. Además, para la especificación de los casos de uso se utilizaron listas de comprobación que garantizaban la presencia de determinadas características deseables en dichas especificaciones. Tanto las métricas como las listas de comprobación verificaban un conjunto de características deseables que propone el modelo de calidad de Davis.
  - La aplicación de estas técnicas permitió verificar la calidad de los requisitos.
  - Para la validación se utilizó la técnica del walkthrough combinado con los prototipos de interfaz de usuario por ser estas las que más se adapta a las necesidades de validación (Durán, 2000.).

Conclusiones generales

- Con la aplicación de las técnicas de elicitación se puede identificar el lenguaje de modelado adecuado para el proceso de negocio, casos de uso y requisitos, así como las herramientas y metodologías adecuadas. Mediante el estudio se obtuvo como resultado las siguientes selecciones:
    - RUP como metodología de desarrollo de software
    - UML como lenguaje de modelado
    - Visual Paradigm versión 6.4 como herramienta de modelado del sistema, negocio y prototipado.
- Los documentos de especificación de requisitos y casos de uso recogieron la especificación de requisitos obtenidas a partir de las necesidades identificadas.
- Desde el inicio de las actividades inherentes de la ingeniería de requisitos aplicadas, se desarrollaron un conjunto de actividades que garantizaron obtener un control y seguimiento de los cambios en los requisitos obtenidos. Para ello se aplicaron algunas técnicas como la trazabilidad y modelado.
- Con la verificación desarrollada se logró la calidad de los documentos y la realización de los prototipos no funcionales.
- Mediante la validación, se obtuvo la aceptación de los documentos que serán entregados.

Recomendaciones

- Continuar perfeccionando el modelado de sistema mediante la actualización de los cambios que sean necesarios durante las etapas de diseño, implementación y pruebas.
- Realizar el diseño e implementación del sistema a partir del modelado de sistema presentado.
- Profundizar en el estudio de los patrones lingüísticos y de reutilización de requerimientos para lograr especificaciones cada vez más estándares y correctas.
- Profundizar en el estudio de la aplicación de la Ingeniería de Requisitos a los procesos de Gestión de Proyectos en general.

Bibliografía

[ISO/IEC1991]. (n.d.). Software Product Evaluation–Quality.Characteristic and Guidelines for their Use. International Standard.9126–1991, International Organization for Standarazition.

Davis, S. O. (1993). *Identifying and measuring quality in software requirements specification.*Los Alamitos,California.

Acuña, K. B. (2009). *Metodologías De Desarrollo Para Aplicaciones Web.*

Agullo, G. (2004). biblioteca universidad complutense.

Ariza. (agosto 2009). CORPORACION UNIVERSITARIA MINUTO DE DIOS “UNIMINUTO”.

Award Winning UML Tool. Visual Paradigm for UML User's Guide. (2007, agosto 1). Retrieved from http://content.europe.visual-paradigm.com/media/documents_download/vpuml61ug1/vpuml_user_guide.pdf

Cockburn, A. (2001). Writing Effective Use Cases. s.l. Adison- Wesley.

Cutisaca, R. J. (2008). Ingeniería de sistemas.

Durán, A. (2000.). *Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información.* Universidad de Sevilla.

ferguson, j. (2003). *la biblia de c#.*

Fernández, S. Z. (2008). Estudio sobre distintas herramientas que dan soporte a la fase de análisis de requisitos. http://petra.euitio.uniovi.es/~i1773418/HdD/trabajos/INFORME%20I.pdf.

Fornaris, M. S., Dayanis Elvia, A. R., & Eylin, H. L. (enero 2010). Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas de Información Geográfica.

http://vinculando.org/articulos/sociedad_america_latina/propuesta_guia_de_medidas_para_evaluacion_sistemas_informacion.html.

Gallego, J. P. (2006). 4 CAPITULOS DE INGENIERIA DE REQUERIMIENTOS. http://www.scribd.com/doc/270431/Ingenieria-requerimientos.

Guerrero, J. (2009). lenguaje de programacion UML.

Jacobson, I. B. (2000). El Proceso Unificado de Desarrollo de Software. pp. 5-8.

Larman, G. (2004). *UML y Patrones.*

Lilly, S. (1999). Use Case–Based Requirements: Review Checklist.

Olson, p. (2006). PHPdoc.

ÖVERGAARD, G., & Karin, P. (2004). Use Cases: Patterns and Blueprints. Addison Wesley.

Pressman, R. (2005). *Ingeniería de Software, un enfoque práctico.*

Rodriguez, S. C. (Noviembre_Diciembre 2005). SG Software Guru.

Seco, j. A. (2006). *El lenguaje de programacion C#.*

souli, s. (2009). programacion C++.

Toro, D. A. (septiembre 2000). Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información.

Toro, D. J. (septiembre de 2000). Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información.

Zapata, C. M. (2008). Esquemas conceptuales UML.

Anexos

Anexo A:

Documento Descripción de los casos de uso del negocio de la Herramienta para la Gestión de Requisitos.

Anexo B:

Documento Requisitos Funcionales de HGR de la Herramienta para la Gestión de Requisitos.

Anexo C:

Documento Requisitos no Funcionales de HGR de la Herramienta para la Gestión de Requisitos.

Anexo D:

Documento Matriz de trazabilidad de la Herramienta para la Gestión de Requisitos.

Anexo E:

Documento Descripción de los casos de uso del sistema de la Herramienta para la Gestión de Requisitos.

Anexo F:

Documento Descripción de los casos de uso del negocio de la Herramienta para la Gestión de Requisitos.

Anexo I:

Documento Lista de chequeo de la Herramienta para la Gestión de Requisitos.

Glosario de términos

CASE: acrónimo de Computer Aided Software Engineering (Ingeniería de Software Asistida por Ordenador), son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

IBM: International Business Machines (conocida coloquialmente como el Gigante Azul) es una empresa que fabrica y comercializa herramientas, programas y servicios relacionados con la informática.

PDF: acrónimo del inglés Portable Document Format (formato de documento portátil), es un formato de almacenamiento de documentos, de tipo compuesto (imagen vectorial, mapa de bits y texto). Está especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la presentación final del documento determinando todos los detalles de cómo va a quedar no requiriéndose procesos anteriores de ajuste ni de maquetación.

Programación Orientada a Objetos (POO): es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (propiedades o datos), comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). Expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

UML: acrónimo del inglés Unified Modeling Language (Lenguaje Unificado de Modelado). Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

XML: acrónimo de Extensible Markup Language («lenguaje de marcas ampliable»), es un metalenguaje extensible de etiquetas, aunque no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.