Universidad de las Ciencias Informáticas

Facultad 2

Título: Herramienta para la implementación de técnicas para la gestión de riesgos de seguridad durante el desarrollo del software.

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

Autor(es): Addiel Velazco Padilla.

Yordanis Calvo Rodriguez.

Tutor: Ing. Arianna Pérez Carmenates.

Co-tutor: MSc. Lilian Teresa Castro Mecias

Junio 2015

Pensamiento

"Siempre elijo a una persona perezosa para hacer un trabajo difícil, porque probablemente encontrará una manera fácil para llevarlo a cabo"

Bill Gates.

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Addiel Velazco Padilla	Yordanis Calvo Rodriguez
Firma Autor	Firma Autor
_____	_____
Arianna Pérez Carmenates	Lilian Teresa Castro
Firma Tutor	Firma Tutor

DATOS DE CONTACTO

acarmenates@uci.cu Ing. Arianna Pérez Carmenates, graduada de Ingeniera en Ciencias Informáticas en el año 2008. Pertenece al Centro de TLM donde se desempeña como jefa del departamento de práctica profesional.

ltcastro@uci.cu MSc. Lilian Teresa Castro Mecias, graduada de Ingeniería en Ciencias Informáticas en 2008. Experiencia de 6 años como docente en la Educación Superior y en el desarrollo de software. Miembro del grupo de investigación de Seguridad en Redes y Sistemas.

AGRADECIMIENTOS:

Al Centro de Telemática (TLM), por abrir la línea de investigación sobre seguridad informática.

A la Universidad de las Ciencias Informáticas UCI, por ser el eslabón de todos en esta cadena ininterrumpida de aprehensiones.

A nuestras tutoras, Arianna Pérez Carmenates y Lilian Teresa Castro Mecias por alentarnos a seguir superándonos profesionalmente.

A nuestras familias, por ser parte de este proceso y apoyarnos decididamente, tanto en momentos de augurios, como en épocas difíciles.

A nuestros colegas y amigos de la UCI por su ayuda y orientación, sin la que hubiera sido imposible la culminación de esta investigación.

A los compañeros del grupo 2509 que me han apoyado con sus criterios y opiniones.

A los especialistas que con tanta voluntad y profesionalidad justificaron mi propuesta.

A los que nos ayudaron incondicionalmente les pedimos que no se fijen en el lugar que fueron mencionados ya que todos tienen un asiento en primera fila en nuestra mente y en nuestro corazón.

A todos muchas gracias.

Los autores.

Addiel Velazco Padilla.

Yordanis Calvo Rodriguez.

DEDICATORIA

En memoria a mi abuelo, Perfecto René Padilla Fernández:

Estuvo presente en cada paso que comencé a dar en mis estudios,

fue el guía de mis anhelados deseos como joven,

cubrió de manera especial todas las etapas de desarrollo en mi niñez, juventud y vida estudiantil en la universidad,

deseaba hasta sus últimos momentos que me emprendiera como hombre de bien, formando en mis experiencias los momentos del deber y el cumplir,

me posibilitó en su tiempo sus anécdotas como combatiente y revolucionario,

pero sin lugar a dudas fomentó en mí los principios del respeto y la voluntad, y arraigó lo elemental del hombre, que no importa las veces que uno caiga sino las batallas por lograr, por reorientarme, por triunfar profesionalmente,

a ti, sangre de mi sangre, destello de mis sentimientos dedico desde lo más profundo de mi corazón esta Tesis.

A mi madre, Tania Padilla Pujol:

Estas palabras no podrán contener todo el profundo sentimiento que me abruman desmesuradamente,

estas palabras no justificarán los momentos difíciles que has pasado en la vida,

estas palabras no admiten regaños, ni locuras, ni rencores,

estas palabras son solo un regalo, un detalle para la madre que me ha dado la vida, que sigue cada paso, lo enfrenta con orgullo y satisfacción,

Satisfacción es hoy brindarte la dicha de que veas que todo lo que has hecho por mí, no han sido en vano,

estas palabras son puro agradecimiento, y llegan a ti justo en el momento en que la vida en mí cambia profesionalmente, ahora queda en mí la responsabilidad, la dedicación y la entrega por todo lo que significas y vales.

Addiel Velazco Padilla.

Hay solo una persona que siempre nos mira con los mismos ojos durante toda nuestra vida. Para ella, siempre somos lindos y hermosos,
para ella somos los más inteligentes y talentosos del mundo,
para ella somos casi perfectos, seres carentes de defectos.

A ti mami por ser la luz que me ha guiado hasta éste, el mayor de mis logros,
a ti, por tu persistencia y dedicación inagotables, por esa confianza tan infinita que depositaste en mí,
de la cual nunca habría imaginado que podría existir tanta en el corazón de una sola persona.

Tu voz ha sabido guiarme, levantándome de los peores fracasos, colmándome de la educación necesaria para ser la persona que soy.

En cada una de tus palabras contemplo esa eterna bondad hacia mí, impulsada por ese amor incondicional que te caracteriza diciéndome que no puedo decepcionarte, que no mereces que te falle.

A ti que me diste todo sin pedir nada, te dedico este triunfo, nuestro triunfo.

Yordanis Calvo Rodriguez.

RESUMEN

El proceso de desarrollo de software está constituido por varias etapas a las que se les asignan responsables de acuerdo con las habilidades de cada informático o las necesidades específicas del proyecto. Muchas veces por cuestiones muy variadas se prescinde de expertos en seguridad en los equipos de desarrollo, esto conlleva a que se obvien muchos requerimientos de seguridad que tienen gran importancia para lograr una aplicación confiable y con un riesgo aceptable. Las buenas prácticas de seguridad son un conjunto de acciones que se acometen en el ciclo de vida del desarrollo del software para proporcionarle mayor calidad a este, estas permiten identificar diferentes amenazas y riesgos a los que estará expuesto el proyecto que está en producción permitiendo crear planes y estrategias para enfrentar dichos peligros.

La presente investigación tiene como objetivo incorporar nuevas técnicas en la herramienta Segursoft[1] para ayudar en la valoración de activos, identificación de amenazas y evaluación de riesgos en correspondencia con el fundamento que se exponen en las buenas prácticas para la gestión de la seguridad en el software, propiciando la creación de productos de mayor calidad y confiabilidad.

PALABRAS CLAVE

Segursoft, buenas prácticas de seguridad, análisis y gestión de riesgos de seguridad, desarrollo de software

[1] Herramienta para el análisis y gestión de riesgos de seguridad.

Índice

Índice de Ilustraciones.

Índice de Tablas.

INTRODUCCIÓN.

Las Tecnologías de la Información y Comunicaciones (TIC) han permitido la interconexión entre las personas e instituciones a nivel mundial eliminando barreras espaciales y temporales. Es un hecho que la sociedad se ha vuelto cada vez más dependiente de las nuevas tecnologías, fundamentalmente por el desarrollo, inmediatez y aceleración de nuevos productos que han facilitado y al mismo tiempo complejizado la vida en el universo.

En la medida en que la sociedad es partícipe de los disímiles espacios que aportan las nuevas tecnologías para ganar en información y conocimiento, en todo el mundo muchos desarrolladores tienen como objetivo brindar continuamente soporte a las mismas, siendo cada vez más importante el tema de la seguridad. En los últimos años han aumentado considerablemente los ataques informáticos, de los cuales las compañías comerciales se han convertido en víctimas potenciales. Ejemplo de esto es el año 2012 en el cual el porcentaje de usuarios atacados en Internet alcanzó el 34% con relación al año anterior (1). Según los resultados de una encuesta realizada en el año 2013 por Kaspersky Lab y la compañía analítica B2B International, el 91% de las empresas encuestadas en todo el mundo fueron víctimas de por lo menos un ataque informático al año y el 9% de las compañías fueron víctimas de ataques selectivos (1). Otra investigación realizada por la consultora PwC reportó que la cantidad de incidentes de seguridad a nivel global creció un 48% durante el año 2014, hasta alcanzar los 42.8 millones, lo cual equivale a 117,339 ataques cada día (2). Estos ataques son realizados aprovechando las vulnerabilidades del sistema atacado y frecuentemente estos elevados porcentajes son debidos a debilidades en el diseño y/o implementación del sistema. El uso de buenas prácticas para asegurar el software durante el ciclo de vida de desarrollo favorece la satisfacción del cliente ya que estas constituyen un conjunto de acciones aplicadas mediante técnicas, estándares y metodologías diseñadas con el fin de garantizar una mejor inserción de la seguridad en cada una de las etapas del ciclo de vida de un software.

Un software seguro debe ser diseñado, implementado y configurado para seguir funcionando correctamente ante la presencia de la mayoría de los ataques, fallas o debilidades, tolerando los errores y fracasos que resulten de tales ataques. (3)

El análisis y gestión de riesgos de seguridad es una de las principales estrategias propuestas por (4) para lograr un software seguro, ya que se encarga de identificar, analizar y gestionar algunas de las amenazas que pueden afectar a un software en desarrollo. El Centro Telemática (TLM) perteneciente a la Universidad de las Ciencias Informáticas (UCI) toma muy en serio la aplicación de la seguridad en los software y servicios que produce, ejemplo de esto es la aplicación de un conjunto de prácticas de seguridad como son la realización de análisis de seguridad, análisis estático del código, integración continua y auditorías a bases de datos, entre otros. Pero la evolución en la que se ha visto inmerso en los últimos años lo ha convertido en un Centro más confiable en cuanto al desarrollo de software y a su vez más comprometido con los clientes. Debido a esto se hace conveniente aumentar el nivel de seguridad que se le aplica a los software en producción tomando como premisa el latente aumento de los riesgos a los que están sometidos los programas informáticos actualmente. Para lo cual se hace necesario la incorporación de nuevas técnicas para el análisis y gestión de riesgos de seguridad de los productos que ofrece el Centro.

El análisis de riesgos es llevado a cabo en el Centro TLM mediante la herramienta Segursoft, la cual fue creada para proporcionar un marco de trabajo para el análisis de riesgos de seguridad. Esta facilita de forma paulatina e incremental la realización de un proceso de desarrollo seguro pero no tiene continuidad con las actividades para la gestión de los riesgos. Segursoft realiza un inventario de activos, pero no se utilizan diagramas que representen la relación entre dichos activos para analizar así la repercusión de la afectación de alguno de estos en el resto de ellos. Permite la modelación de árboles de ataque que identifican las diferentes acciones a realizar por un atacante para cumplir un determinado objetivo, pero no se especifica en ellos la importancia de estas acciones para de esta forma poder priorizarlas y planificar las medidas a acometer para evitarlas. Además brinda la posibilidad de modelar diagramas de casos de abuso, los cuales representan las acciones indebidas que se realizan sobre un sistema desde el punto de vista del atacante, pero no ofrece la opción de describir los casos de mal uso, que representan las acciones indebidas realizadas en el sistema y de especificar los requisitos de seguridad necesarios para contrarrestarlos aumentando la seguridad del software en desarrollo. Esta herramienta con sus actuales prestaciones no satisface en su totalidad uno de los principales pilares para la seguridad del software, el análisis y gestión de riesgos, ni las necesidades actuales del Centro TLM debido a que este gestiona información sensible que de ser manipulada indebidamente puede llegar a causar daños a la universidad y a sus clientes.

De lo cual se deriva el **problema a resolver:** Las limitaciones presentes en la herramienta Segursoft para la aplicación de técnicas para el análisis y gestión de riesgos de seguridad disminuye la capacidad del Centro TLM para asegurar el software.

Definiendo como **objeto de estudio:** Buenas prácticas de seguridad en el desarrollo de software.

El **objetivo general** de esta investigación es: Incorporar en Segursoft nuevas técnicas de análisis y gestión de riesgos de seguridad que permitan superar las limitaciones que presenta, aumentando la capacidad del Centro TLM para asegurar el software y como **campo de acción**: Técnicas para el análisis y gestión de riesgos de seguridad.

Para dar cumplimiento a los objetivos se precisan las siguientes **tareas de la investigación:**

1. Investigación y estudio de las herramientas de seguridad existentes que implementen técnicas para el análisis y gestión de riesgos de seguridad, para reunir información acerca del estado actual de estas, su diseño y características básicas, de las cuales se tomarán aportes para implementar la solución propuesta.
2. Estudio de los fundamentos del análisis y gestión de riesgos de seguridad en el desarrollo del software para su posterior aplicación en la versión 2.0 de la herramienta Segursoft.
3. Estudio de las técnicas de evaluación de riesgos y especificación de requisitos de seguridad para decidir cuáles adicionar y cómo incorporarlas en la nueva versión propuesta.
4. Análisis de las metodologías, herramientas y tecnologías para el diseño y desarrollo de la nueva versión.
5. Estudio de los diferentes tipos de pruebas realizadas a sistemas informáticos como Caja Blanca y de Caja Negra para definir cuál aplicar a la solución propuesta.

Idea a defender:

Desarrollando la versión 2.0 de la herramienta Segursoft quedan resueltas las limitaciones presentes en dicha herramienta en cuanto a la aplicación de nuevas técnicas para el análisis y gestión de riesgos de seguridad aumentando la capacidad del Centro TLM para asegurar el software durante su proceso de desarrollo.

Para el desarrollo de la investigación se hace necesario el empleo de los siguientes métodos de la investigación:

Métodos Teóricos:

Análisis y síntesis: Se usa el análisis a lo largo de toda la investigación realizada acerca de la aplicación de técnicas para el análisis y gestión de riesgos de seguridad con el fin de obtener todos los datos necesarios acerca de estas buenas prácticas y luego mediante la síntesis obtener una organizada y detallada información con el fin de utilizarla en el desarrollo de la nueva versión que solucione el problema existente.

Histórico-lógico: Se utiliza este método a lo largo de toda la investigación y el estudio realizado acerca de la herramienta Segursoft. Mediante este método se investiga cómo ha sido el desarrollo histórico de la herramienta y cómo se ha comportado esta desde su creación a la vez que se ha estudiado la lógica de su desarrollo y que elementos son esenciales en esta herramienta.

Métodos Empíricos:

Observación.

En la presente investigación se utiliza este método a la vez que se instalan varias de las herramientas existentes relacionadas con la seguridad en las diferentes etapas de desarrollo del software, para estudiar de ellas las principales características y para identificar deficiencias o limitaciones en estas herramientas que podrían ser evitadas en la solución propuesta. Además para la observación del proceso de desarrollo de software que se sigue en el Centro que permitirá conocer las deficiencias existentes en la gestión de la seguridad.

Entrevista.

En la investigación se utiliza la entrevista para extraer los diferentes procesos de negocio y funcionalidades que debe presentar la solución a implementar, estos datos son aportados por la especialista que propuso la necesidad de la nueva versión de la herramienta Segursoft. Se realiza mediante encuentros que pueden se planificados o no, en los que se discute y valora el avance de la solución que se implementa. Esta entrevista no es estructurada ya que no consta de un cuestionario fijo de preguntas.

El contenido de la investigación se encuentra distribuido de la siguiente manera:

Capítulo 1. **Fundamentación teórica:**

Se plasma el estudio del estado del arte del objeto de estudio que fundamenta la investigación y se muestran los elementos teóricos correspondientes a la solución propuesta, entre los que se encuentran los conceptos necesarios para un mejor entendimiento del tema de la seguridad informática como son el análisis y gestión de riesgos de seguridad y varias de las técnicas que se utilizan para implementarlo. Además se describe el estudio realizado a un conjunto de herramientas y metodologías que implementan diferentes técnicas y procedimientos para facilitar la aplicación de la seguridad al software desde las etapas iniciales de su desarrollo. Por último se mencionan las diferentes tecnologías a usar para desarrollar la solución que se propone.

Capítulo 2. **Exploración y planificación:**

En este capítulo se exponen las funcionalidades que debe cumplir la solución propuesta, además se describen las características de la misma y se hace una descripción de la arquitectura de la propuesta a desarrollar.

Capítulo 3. **Diseño, Implementación y prueba:**

En este capítulo se describen los principales elementos del diseño de la propuesta, se realiza la implementación de la misma y se describen los procedimientos necesarios para su desarrollo. Se confecciona el diseño de las pruebas a realizar y se aplican dichas pruebas para comprobar el funcionamiento de la solución.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En el mundo la necesidad tecnológica se hace cada día más frecuente, debido a que la sociedad es cada vez más dependiente de los nuevos cambios tecnológicos. Ligada a ésta la manipulación de información y datos juega un papel primordial ya que las aplicaciones están cada vez más expuestas a peligros en la red. En el presente capítulo se describen diferentes conceptos referentes a la seguridad aplicada en las etapas del desarrollo de un software haciendo énfasis en el análisis y gestión de riesgos de seguridad, se explican sus bases y se detalla un conjunto de técnicas que lo implementan. Además se hará un estudio del estado del arte en el que se describen sistemas semejantes para tomar de ellos los aportes y técnicas que por sus características se consideren útiles. Por último se describen las diferentes tecnologías, metodologías y herramientas utilizadas para desarrollar la solución que se propone.

1.2 Seguridad en el proceso de desarrollo de software.

La seguridad debe estar presente en todo el ciclo de vida del software. Un software seguro es creado de manera que cumpla con las propiedades esenciales, las cuales son, que continúe operando en presencia de ataques, aísle o limite los daños y se recupere lo antes posible. (5) McGraw plantea que la seguridad del software se basa en tres pilares fundamentales (4): la administración del riesgo, la aplicación de buenas prácticas en etapas del ciclo de vida de desarrollo y el conocimiento, como se muestra en la Ilustración 1.

Ilustración 1 Pilares fundamentales en el desarrollo de software seguro.
Fuente: (4)

La seguridad de un software no debe consistir solo en proteger dicho programa una vez construido ni el hecho de establecer contraseñas y derechos de acceso o enmendar las fallas de seguridad que surjan mientras ya está en uso. Para propiciarle al cliente un software seguro es necesario el análisis de sus vulnerabilidades y posibles amenazas a lo largo del ciclo de desarrollo haciendo énfasis en cada una de sus etapas logrando así prever daños que pueden ser irreparables tanto para la aplicación como para el usuario que la utiliza, en la Ilustración 2 se observa un conjunto de buenas prácticas a seguir para incorporar la seguridad en todas la etapas de desarrollo. La seguridad no puede verse solo como una característica más de la aplicación esta debe considerarse como una propiedad emergente de un sistema de software e incorporarse desde el principio del desarrollo. (6) Este es un tema que actualmente ha tomado auge en la comunidad informática y es por su importancia que varios investigadores y especialistas en el tema proponen modelos, metodologías de desarrollo y la implementación de buenas prácticas para su inserción en un proceso de desarrollo de software mejorado para la seguridad.

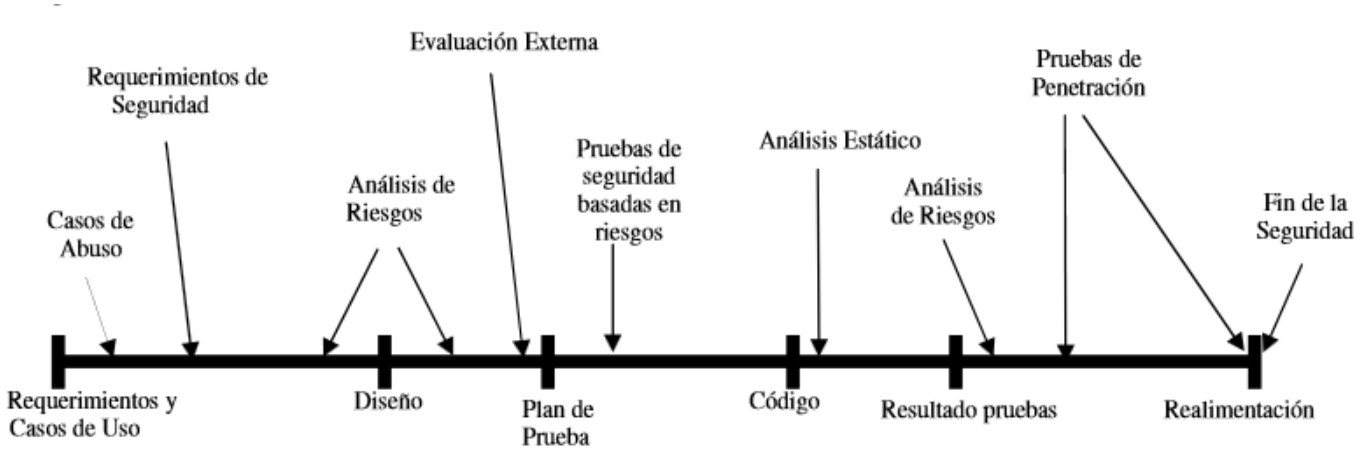


Ilustración 2 Seguridad en todas las etapas de desarrollo.
Fuente: (7)

1.3 Buenas prácticas en el proceso de desarrollo de software.

Las buenas prácticas también constituyen un pilar fundamental en la creación de software seguro y establecen medidas que posibilitan hacer uso de metodologías y estándares de seguridad en las diferentes etapas de desarrollo. McGraw plantea en (4) que las buenas prácticas no son un proceso de desarrollo, el las considera como prácticas agnósticas, o sea que se pueden aplicar indistintamente al modelo de ciclo de

vida que se utilice. Pretenden ofrecer a los desarrolladores un camino a seguir hacia su incorporación desde las primeras etapas del ciclo de vida, para lograr desarrollos de software seguros. (6)

1.4 Análisis y gestión de riesgos de seguridad.

Análisis de riesgos.

El análisis de riesgos no es más que un proceso sistemático para estimar la magnitud de los riesgos a que está expuesta una Organización, este permite determinar cómo es, cuánto vale y cómo de protegido se encuentra el sistema. El análisis de riesgos es una aproximación metódica para determinar el riesgo siguiendo unos pasos pautados (8):

1. Determinar los activos relevantes para la Organización, su interrelación y su valor, en el sentido de qué perjuicio (coste) supondría su degradación.
2. Determinar a qué amenazas están expuestos aquellos activos.
3. Determinar qué salvaguardas hay dispuestas y cuán eficaces son frente al riesgo.
4. Estimar el impacto, definido como el daño sobre el activo derivado de la materialización de la amenaza.
5. Estimar el riesgo, definido como el impacto ponderado con la tasa de ocurrencia (o expectativa de materialización) de la amenaza.

Además de los pasos anteriormente mencionados la herramienta Segursoft implementa otras técnicas para el análisis de riesgos, ejemplo de ello es el proceso de modelado de amenazas el cual se realiza a través de varios diagramas que representan las posibles amenazas y las vías por la cuales se pueden materializar.

Gestión de riesgos.

A partir del análisis de riesgos se dispone de información para tomar decisiones conociendo lo que queremos proteger (activos valorados), de qué lo queremos proteger (amenazas valoradas) y qué hemos hecho por protegerlo (salvaguardas valoradas). (8) Estas decisiones no son más que el proceso de gestión de riesgos. En (4) se argumenta que la aplicación de un enfoque de gestión de riesgos a lo largo de todo el

ciclo de vida, es una filosofía que asegura todo el trabajo que se realice sobre la seguridad del software. La idea básica es identificar, clasificar y priorizar los riesgos de seguridad de software mientras se aplican las buenas prácticas durante todo el ciclo de vida.

Las tareas de análisis y tratamiento de los riesgos no son un fin en sí mismas sino que se encajan en la actividad continua de gestión de la seguridad. Para el proceso de análisis y gestión de riesgos se han desarrollado varias metodologías y técnicas con el fin de estandarizar y guiar el trabajo a los usuarios que realicen dicho proceso.

1.4.1 Técnicas para el análisis y gestión de riesgo.

Técnica para el cálculo del nivel de riesgo propuesta por MAGERIT.

Una de las metodologías desarrolladas para llevar a cabo el análisis y gestión de riesgos de seguridad es la Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información (MAGERIT) la cual propone un método para la toma de decisiones con fundamentos. Esta sugiere una técnica que permite determinar el nivel de riesgo que supone una determinada amenaza basándose en el impacto de la misma (6), dicha técnica se enfoca en el paso número 5 del proceso de análisis de riesgos. El modelo de estimación de riesgo de una amenaza que propone MAGERIT tiene en cuenta dos criterios (el impacto y la probabilidad de ocurrencia) (9), pero permite a la vez obtener como salida a este cálculo dos nuevas clasificaciones (muy alto y muy bajo) a diferencia de la técnica DREAD implementada por Segursoft que solo muestra como salida (bajo, medio y alto). DREAD es una técnica para estimar el nivel de riesgo de una amenaza, esta debe su nombre a la traducción al inglés de los 5 parámetros en los que se basa para realizar dicha estimación (Reproducibilidad, Explotabilidad, Daño potencial, Descubrimiento y Usuarios afectados). Debido a la simplicidad del análisis que ofrece MAGERIT y a la profundidad y nivel de detalle de los resultados que presenta como salida de este cálculo se hace conveniente brindar al usuario la posibilidad de hacer uso de la técnica que propone esta metodología para estimar el nivel de riesgo de una amenaza.

Para el análisis y gestión de riesgos de seguridad le serán incorporados a la herramienta Segursoft un conjunto de nuevos diagramas los cuales son los grafos de ataques, los casos de mal uso y los diagramas de activos. También se le incorporarán dos nuevas técnicas que permitirán la descripción de casos de mal uso y la especificación de requisitos de seguridad a partir de los casos de uso de seguridad.

Grafos de ataques.

Estos pertenecen al proceso de modelado de amenazas y consisten en una representación de todos los caminos a través de un sistema que acaban en un estado en el que un intruso o atacante ha conseguido de forma satisfactoria su objetivo. En los grafos de ataque, los nodos representan el estado en el que se encuentra un ataque y las aristas representan un determinado ataque o una transición entre estados de un ataque (10). Mientras los árboles de ataque modelan el plan de actuación del atacante el grafo de ataque modela el progreso del atacante. Esta orientación hacia el objetivo hace que los grafos sean unidireccionales o acíclicos. Al igual que los árboles, hay diferentes formas de llegar a un cierto punto intermedio, mediante formas alternativas (OR) o formas que requieren la conjunción de éxitos (AND) (9).

Este es un método flexible en el sentido de que permite definir de la misma forma un variado conjunto de ataques complejos. Además, permite tanto el análisis de los ataques provenientes de redes exteriores como de redes interiores. Pueden analizar el riesgo de un elemento específico de una red o examinar todas las posibles consecuencias provocadas por un ataque que ha tenido éxito. El inconveniente que tienen los grafos de ataque es que cada uno de ellos define de una forma fija la serie de pasos que un atacante debería seguir para realizar una acción peligrosa contra un sistema. Esto hace que cuando un ataque varíe uno de los pasos el grafo no funcione como cabría esperar. Cada arista o transición de un grafo de ataque tiene un peso, el cuál presenta la probabilidad de éxito o el tiempo medio para suceder. Las transiciones etiquetadas con un peso de 0 son, por lo general, omitidas. Un camino corto o pequeño en el grafo de ataque representa un ataque de un coste pequeño. (10)

Diagrama de casos de mal uso.

Los diagramas de casos de mal uso pertenecen al proceso de modelado de amenazas para representar la perspectiva de un usuario que hace mal uso de los permisos que le son asignados en un sistema. Estos aplican el concepto de un escenario negativo, una situación que el dueño del sistema no quiere que se produzca, en el diagrama de casos de mal uso a dichas situaciones se les llama casos de mal uso y le son asignados casos de uso de seguridad los cuales especifican las acciones a acometer para contrarrestarlas. (11) A continuación se muestra una tabla comparando los casos de mal uso y los casos de uso de seguridad en cuanto a un grupo de criterios definidos en (12).

Tabla 1 Comparación entre los casos de mal uso y los casos de uso de seguridad.

	Caso de mal uso	Caso de uso de seguridad
Utilización	Analizar y especificar amenazas de seguridad	Analizar y especificar requisitos de seguridad
Criterios de éxitos	Mal usuario tiene éxito	Tiene éxito la aplicación
Creado por	Equipo de seguridad	Equipo de seguridad
Usado por	Equipo de seguridad	Equipo de requerimientos
Actores	Usuario, Mal usuario	Usuario
Conducido por	Análisis de amenazas Análisis de vulnerabilidades de activos	Casos de mal uso

Los casos de mal uso no son un simple método de modelado, sino que proporcionan oportunidades para investigar y validar los requisitos de seguridad necesarios para llevar a cabo la misión del sistema. Esto se debe a que una de las técnicas creadas para la gestión de riesgos de seguridad es la descripción de casos de mal uso y la especificación de los requisitos de seguridad mediante la descripción de casos de uso de seguridad pertenecientes a los casos de mal uso. Los casos de mal uso se concentran en las interacciones entre la aplicación y los atacantes que tratan de violar su seguridad. Debido a que los criterios de éxito para un caso de mal uso es un ataque exitoso contra una aplicación, los casos de mal uso son formas muy eficaces de análisis de amenazas a la seguridad, pero no son adecuadas para el análisis y la especificación de los requisitos de seguridad. En cambio, los casos de uso de seguridad deben ser usados para especificar los requisitos de seguridad que deberán proteger con éxito al sistema de sus amenazas de seguridad correspondientes (casos de mal uso). Los requisitos de seguridad deben basarse en un análisis de los activos y servicios a ser protegidos y las amenazas a la seguridad de las que estos bienes y servicios deben ser protegidos. (12)

Para describir casos de mal uso y especificar requisitos de seguridad han surgido diferentes técnicas como la creación de plantillas que recogen un conjunto de elementos necesarios para facilitar y estandarizar el proceso. En la Tabla 63 se muestra un ejemplo de la plantilla propuesta por (11) para llevar a cabo la técnica de descripción de casos de mal uso y en la Tabla 64 se describe la plantilla propuesta por (12) para la especificación de los requisitos de seguridad mediante los casos de uso de seguridad.

1.5 Estudio del estado del arte.

En la actualidad existe gran diversidad de herramientas creadas para ayudar en la medición y mitigación del riesgo inherente al desarrollo de software y facilitar así la creación de sistemas más seguros. Dentro de estos sistemas destacan:

1.5.1 Microsoft Threat Analysis and Modeling Tool v3.0.

Microsoft Threat Analysis and Modeling Tool (TAM, por sus siglas en inglés) automáticamente identifica las amenazas de un proyecto y además produce los artefactos de seguridad. Esta posee una biblioteca de ataques con una guía descriptiva de contramedidas para cada uno. También permite la navegación usando el componente treeview pudiendo visualizar todos los nodos expandidos de forma simultánea. Esta genera informes y estadísticas exportables a HTML y posee video tutoriales. Posee una superficie de dibujo de Visio donde los usuarios pueden arrastrar y soltar funciones y componentes para construir un caso de uso. (13)

A partir de los requerimientos y la descripción de la arquitectura, la herramienta trata de identificar de manera automática las amenazas, al tiempo que produce una serie de elementos como son:

- Matrices de acceso a datos.
- Casos de uso.
- Diagramas de flujos de datos, de llamada, y de confianza.
- Superficies de ataque.

Cabe destacar que esta herramienta solo es utilizable en la plataforma Windows lo cual imposibilita su uso en la universidad debido a la migración hacia sistemas Linux en la que se encuentra inmersa actualmente, además tiene licencia limitada lo que imposibilita a usuarios que utilicen la plataforma Windows hacer pleno uso de sus versiones y funcionalidades.

1.5.2 Consultative Objective Risk Analysis System.

Consultative Objective Risk Analysis System (CORAS, por sus siglas en inglés) es un proyecto creado por la unión europea con el objetivo de proporcionar un framework orientado a sistemas donde la seguridad es

crítica, facilitando el descubrimiento de vulnerabilidades de seguridad, inconsistencias y redundancias. Esta proporciona un método basado en modelos, para realizar análisis de riesgos, y se basa en el uso de dos componentes (14):

- La metodología CORAS, la cual es una descripción detallada del proceso de análisis con una directriz para construir diagramas.
- Una herramienta para documentar, mantener y crear los informes del análisis.

Esta metodología hace un uso intensivo de los diagramas. Existen 5 tipos diferentes:

Diagrama superficial de activos: Muestra una visión general de los activos y cómo el daño sobre un activo puede afectar al resto. Debido a que los activos son los componentes o funcionalidades de un sistema de información (información, datos, servicios, aplicaciones (software), equipos (hardware), comunicaciones, recursos administrativos, recursos físicos y recursos humanos), al ser atacado alguno deliberada o accidentalmente, conllevaría altas consecuencias para la organización. (8) En la Ilustración 3 se puede observar un ejemplo de un diagrama de activos.

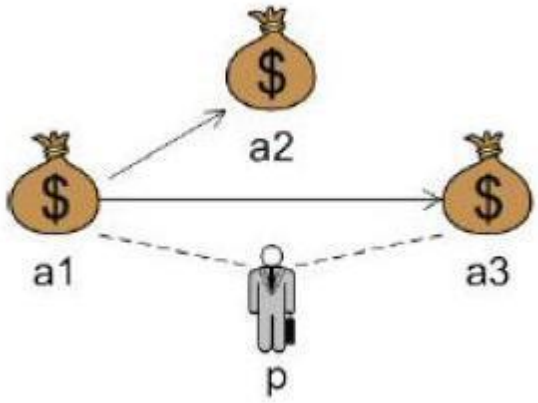


Ilustración 3 Diagrama superficial de activos
Fuente: (14)

Diagrama de amenazas: Muestra una visión completa de la secuencia de eventos iniciados por las amenazas y las consecuencias que tienen éstas sobre los activos. Sus componentes básicos son: amenazas deliberadas, amenazas accidentales, amenazas no-humanas, vulnerabilidades, escenarios de

amenazas, incidentes no deseados y activos. La Ilustración 4 muestra un ejemplo de un diagrama de amenazas.

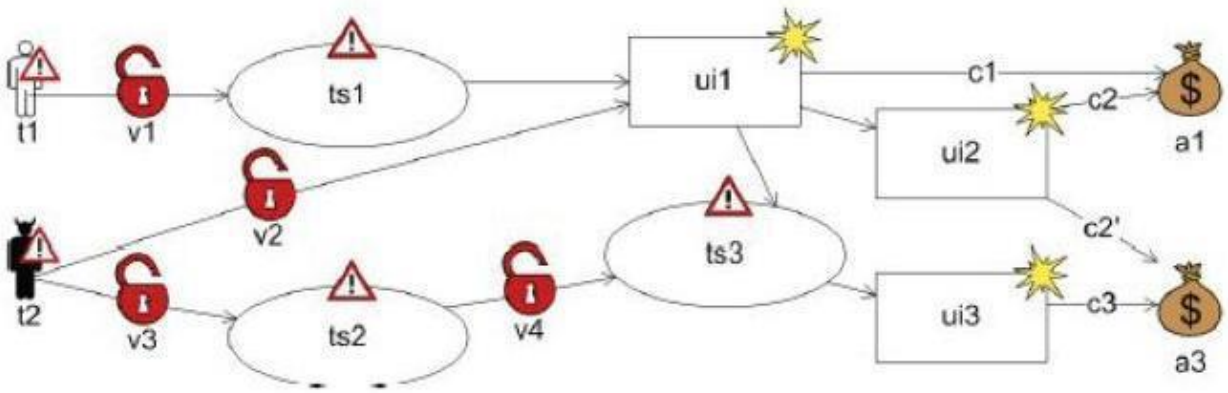


Ilustración 4 Diagrama de amenazas
Fuente: (14)

Diagrama superficial de riesgo: Es un resumen del diagrama de amenazas, tiene 5 componentes básicos: amenazas deliberadas, accidentales y no-humanas, riesgos y activos. A cada riesgo se le asigna un valor.

Diagrama de tratamiento: Ofrece una visión completa de las contramedidas propuestas. Se basa en el diagrama de amenazas, sustituyendo las consecuencias del impacto sobre los activos con los riesgos procedentes del diagrama superficial de riesgo y añadiendo los escenarios de contramedidas propuestos.

Diagrama superficial de tratamiento: Es un resumen de las contramedidas, añadiendo los distintos escenarios posibles y mostrando las relaciones entre los distintos elementos propuestos para tratar el riesgo.

La aplicación principal permite crear nuevos proyectos de análisis, editar resultados de los análisis, generar informes así como reutilizar información procedente de otros análisis. Utiliza dos bases de datos a modo de repositorios. El repositorio de evaluación almacena todos los resultados procedentes del análisis, mientras que el repositorio de experiencia, contiene resultados reutilizables procedentes de anteriores análisis como modelos UML, procedimientos, o listas de comprobación. (14)

A pesar de las características antes mencionadas, la documentación relacionada con esta herramienta es escasa. La herramienta no incluye uno de los principales pasos de la técnica de modelado de amenazas,

que es la descomposición de la aplicación, para lograr conocer en profundidad su arquitectura y el diseño identificando los roles de usuario, las dependencias externas y componentes de la aplicación, lo que imposibilita analizar con mayor profundidad las posibles debilidades del software que se analiza. Pero conociendo las posibilidades que nos brinda CORAS con respecto al análisis de los activos se incorporará uno de los 5 diagramas utilizados por esta metodología a la nueva versión que se pretende realizar a la herramienta Segursoft, el diagrama superficial de activos, para suprimir de esta forma la limitación presente en dicha herramienta con respecto al modelado de la relación entre los activos que se gestionan en los proyectos de análisis.

1.5.3 Practical Threat Analysis 1.6.

Practical Threat Analysis (PTA, por sus siglas en inglés) está basada en la metodología de evaluación de riesgos PTA y permite a los consultores de seguridad y los usuarios de la organización encontrar una manera beneficiosa y rentable para proteger los sistemas y aplicaciones de acuerdo a su funcionalidad y entorno específico. Propone una suite para la elaboración de modelos de gestión de riesgos y permite estimar un nivel de seguridad a partir de la información incluida en cada proyecto. Además de recomendar las medidas más rentables, esta herramienta presenta el actual nivel de seguridad del sistema monitorizado. Una vez utilizada, permite cambios dinámicos en cada una de las amenazas, vulnerabilidades, activos definidos y los parámetros de contramedidas. Esto permite una gestión eficaz y continua de evaluación de los riesgos y la seguridad a través de la rutina de los negocios sin duplicar esfuerzos y con un coste mínimo. (15)

La herramienta PTA permite profundizar también en el detalle de los distintos elementos que conformarán el modelado. También permite la importación automática de datos de entidades y sus parámetros desde fuentes externas como escáneres de vulnerabilidades. Los distintos métodos de cálculo se pueden adaptar a diferentes situaciones, por ejemplo los valores financieros que se asignan a los distintos activos se pueden calcular utilizando diferentes fórmulas. Utiliza librerías de entidades, permite elaborar listas de verificación en conformación con diferentes estándares de seguridad como ISO 17799 y BS7799 y permite la elaboración de un variado número de informes. (14)

Esta herramienta solo es compatible con el sistema operativo Windows en las versiones (15):

- Windows XP

- Windows Server 2003 + SP1
- Windows Server 2008 + SP1
- Windows Vista Ultimate + SP1
- Windows 7

Por tal motivo debido a la migración en la que se encuentra inmersa la universidad es poco factible hacer uso de esta herramienta.

1.6 Tecnologías para el desarrollo.

1.6.1 Metodología.

Actualmente existen numerosas propuestas de metodologías que intervienen de distintas formas en el proceso de desarrollo del software. Estas metodologías son marcos de trabajo que forman parte de un conjunto de procedimientos y técnicas usadas para estructurar, planificar y controlar este proceso generando la documentación necesaria para el mismo.

Dentro de estas metodologías existen dos grandes grupos, las metodologías tradicionales o pesadas y las ágiles. Las tradicionales o pesadas, dentro de las que se encuentra Rational Unified Process (RUP, por sus siglas en inglés) se caracterizan por hacer énfasis en la planificación total del trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada, mientras que las metodologías ágiles, dentro de las que se encuentra Xtreme Programming (XP, por su siglas en inglés) entre otras, enfatizan las comunicaciones cara a cara en vez de la documentación por estar especialmente orientadas para proyectos pequeños, constituyendo una solución a medida para ese entorno y aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones

implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos. Debido a que establece que la programación debe ser en parejas y en constante comunicación con el cliente mejora la productividad de los proyectos y garantiza la calidad del software desarrollado, haciendo que este supere las expectativas del cliente. (16)

Dadas estas características se decide utilizar XP para el desarrollo del presente trabajo de diploma ya que es un proyecto pequeño donde todo el trabajo es realizado por una pareja de programadores y la entrega es en un corto tiempo. Los requisitos pueden cambiar conforme avanza el trabajo y esta metodología plantea que el cliente puede agregar nuevas Historias de Usuario (HU), dividirlas o simplemente eliminarlas. El cliente además forma parte del equipo de desarrollo logrando una mejor retroalimentación, y con la abundante documentación de la versión anterior contribuye a un mejor desempeño del equipo, mejorando además la corrección de errores y permitiendo obtener un producto que satisfaga todas sus necesidades.

1.6.2 Visual Paradigm 8.0.

Herramientas Case: Constituyen un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un sistema. (17)

Dentro de la familia de las herramientas CASE, Visual Paradigm es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software. Permite dibujar todos los tipos de diagramas de clases y presenta licencia gratuita y comercial. Es fácil de instalar, actualizar y es compatible entre sus distintas versiones. Esta herramienta posibilita diseñar, integrar, automatizar y monitorear los procesos del negocio por medio de un ambiente gráfico, el cual es fácil de aprender. (18) En la presente investigación se utiliza la herramienta Visual Paradigm en su versión 8.0, para modelar el diagrama Entidad Relación correspondiente a la solución propuesta.

1.6.3 Entorno de desarrollo: NetBeans 7.3.

NetBeans es un IDE (Integrate Development Environment, por sus siglas en inglés), que al español se traduce como entorno de desarrollo integrado. Este IDE de desarrollo es de código abierto y multiplataforma. Posee un sistema para examinar todos los directorios de cada proyecto, haciendo reconocimiento y carga de clases, métodos y objetos, para acelerar la programación. (19)

1.6.4 Lenguaje de programación: Java.

Para la implementación de la nueva solución se empleó Java como lenguaje de programación, por ser el lenguaje utilizado en la herramienta Segursoft en su versión 1.0. Java es un lenguaje de programación orientado a objeto y de plataforma independiente. El lenguaje toma sintaxis de lenguajes como C y C++, aunque tiene un modelo de objeto más simple y elimina herramientas de bajo nivel. Sus características lo han llevado a niveles muy altos en la preferencia de los desarrolladores de la comunidad internacional. (20)

Es un lenguaje multiplataforma, por lo que la herramienta a desarrollar podrá ser ejecutada en cualquier sistema operativo. Provee un grupo de componentes visuales que agilizan el trabajo del desarrollador como por ejemplo tablas y el Jtree, los cuales pueden ser utilizados y personalizados. Además proporciona el uso de distintas librerías que serán de utilidad para el desarrollo de la herramienta, como es el caso de Itext que permite la generación de documentos en formato pdf e Hibernate que posibilita el mapeo relacional y almacenar información en bases de datos.

1.6.5 Hibernate.

Esta es una capa de persistencia objeto/relacional y un generador de sentencias sql. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada permite generar BBDD en cualquiera de los entornos soportados: Oracle, DB2, MySql, etc. Además es open source, lo que supone, entre otras cosas, que no es necesario pagar por adquirirlo. (21) Su arquitectura base se muestra en la Ilustración 5.

Ilustración 5 Arquitectura base.
Fuente: Elaboración propia

Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL, busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Este genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución. También tiene la funcionalidad de crear la base de datos a partir de la información disponible, ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria"). (22)

1.6.6 Microsoft Visio.

Es un software de dibujo vectorial para Microsoft Windows que permite realizar diagramas de oficinas, diagramas de bases de datos, diagramas de flujo de programas, UML y más. Posee la capacidad de simplificar un diagrama grande y complejo. Presenta herramientas específicas para prácticamente cualquier trabajo de documentación de procesos empresariales y diagramas de flujo adicionales, como IDEF0

(Definición de integración para modelo de proceso) el cual será usado para modelar los procesos de negocio del sistema. (23)

1.6.7 IDEF0.

IDEF0 es una técnica de modelación concebida para representar de manera estructurada y jerárquica las actividades que conforman un sistema o empresa, y los objetos o datos que soportan la interacción de esas actividades. Un modelo IDEF0 se compone de una serie jerárquica de diagramas que permiten mediante niveles de detalle, describir las funciones especificadas en el nivel superior. En las vistas superiores del modelo la interacción entre las actividades representadas permite visualizar los procesos fundamentales que sustentan la organización. Los elementos gráficos utilizados para la construcción de los diagramas IDEF0 son cuadros y flechas como se muestra en la Ilustración 6. (24)

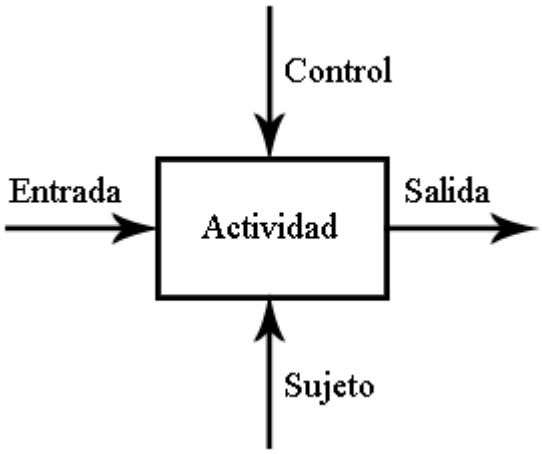


Ilustración 6 Elementos gráficos de la técnica IDEF0.
Fuente: (24)

Ventajas de la técnica IDEF0.

- Permite representar el proceso cronológicamente y se describe el flujo orientado al cliente final del negocio.
- Es una notación simple (basada en cuadros y flechas) que se puede usar para describir qué debe hacer el usuario en el negocio.

- Permite incorporar en el flujo los datos que entran y salen de las actividades, así como las reglas del negocio y los actores, todo en la misma vista.
- Permite descomponer una actividad como un proceso a su vez.

Dadas estas características y la simplicidad que brinda esta técnica se decide utilizarla para modelar los procesos de negocio, asegurando un mejor entendimiento del flujo de las actividades pertenecientes a la nueva versión a implementar.

1.6.8 PostgreSQL 9.1.

PostgreSQL es un Sistema de gestión de bases de datos relacional orientado a objetos, publicado bajo la licencia BSD y con su código fuente disponible libremente. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. Este es fácil de administrar y su sintaxis SQL es estándar. Es bastante poderoso con una configuración adecuada, multiplataforma y cuenta con soporte empresarial disponible. Este utiliza un modelo cliente/servidor y usa multiprocesos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. (25)

1.6.9 Spring Framework.

En ocasiones para desarrollar una aplicación no es suficiente con usar un único framework sino que es necesario utilizar varios de estos, pero los cuales son totalmente independientes y gestionan su propio ciclo de vida de los objetos. Spring ayuda a resolver este problema ya que cambia las responsabilidades y en vez de que el propio desarrollador sea el encargado de generar los objetos de cada uno de los frameworks es Spring basándose en ficheros XML o anotaciones el encargado de construir todos los objetos que la aplicación va a utilizar. (26) Spring proporciona además varios módulos que abarcan la mayor parte de las tareas que se deben hacer en cualquier capa de una aplicación.

Ventajas del uso de Spring:

- Permite que se maneje la infraestructura de la aplicación, por lo que los desarrolladores solo se deben preocupar de la lógica del sistema y de la configuración de Spring.

- Para utilizar Spring no es necesario extender ni implementar alguna clase o interfaz de Spring si no se desea, por lo que el código queda libre y completamente reutilizable para un proyecto sin Spring.
- Permite que la aplicación quede dividida en capas bien delimitadas y con buenas prácticas de programación.
- Debido a que el núcleo de Spring está basado en el patrón de diseño Inversión de Control (IC por sus siglas en inglés), inversión significa que la aplicación no controla su estructura permite que sea Spring quien lo haga.
- Posibilita la inyección de dependencia en tiempo de ejecución, donde los componentes no saben cuál implementación concreta de otros componentes están usando, solo ven sus interfaces.

1.7 Conclusiones parciales.

En el presente capítulo luego de la investigación realizada se han plasmado los principales conceptos referentes a la seguridad del software. Los mismos son aplicados a través de varias técnicas diseñadas para implementar buenas prácticas de seguridad en las diferentes etapas del desarrollo de aplicaciones, entre las que se encuentra el análisis y gestión de riesgos de seguridad. Luego de un estudio de sistemas semejantes a la herramienta Segursoft y un conjunto de metodologías y técnicas creadas para implementar el análisis y gestión de riesgos de seguridad, se han seleccionado un grupo de nuevas características y funcionalidades que pueden ser agregadas a esta herramienta, con el fin de mejorar los servicios que presta y permitir una mejor incorporación de la seguridad en las etapas iniciales del ciclo de vida de los productos que se desarrollan en el Centro TLM. Luego de una comparación entre las diferentes metodologías de desarrollo y teniendo en cuenta las características de la presente investigación se seleccionó la metodología XP como la más indicada para guiar la implementación de la solución propuesta. Finalmente, se realizó una investigación de las herramientas y tecnologías necesarias para llevar a cabo la implementación de las nuevas funcionalidades en Segursoft, para comprobar la existencia de versiones superiores a las que fueron empleadas en la creación de esta herramienta y facilitar el desarrollo de su segunda versión.

CAPÍTULO 2: EXPLORACIÓN Y PLANIFICACIÓN.

2.1 Introducción.

En el presente capítulo se realiza una descripción de las etapas que presenta la metodología XP referentes a las fases de exploración y planificación de la solución propuesta destacándose los principales artefactos que son generados en estas fases. Entre los artefactos que se generarán se encuentran las HU las cuales plasman los requerimientos descritos por el cliente de forma superficial, el plan de iteraciones en el que se mencionan las HU asociadas a cada iteración y el plan de entrega para definir las fechas de terminación de las versiones funcionales que serán entregadas al cliente después de cada iteración. Se describe también el modelo arquitectónico para tener definida la estructura del sistema facilitando así la implementación del mismo.

2.2 Segursoft v1.0.

Segursoft en su versión 1.0 es una herramienta que fue creada con el objetivo de gestionar la identificación de riesgos durante el proceso de desarrollo de software a través del análisis de riesgos de seguridad, siendo capaz de proporcionar un mecanismo que cumpla con cada uno de los pasos que propone la técnica. La herramienta permite que se gestionen los activos más importantes de la aplicación que se modela, así como gestionar los roles de usuarios, principales componentes y dependencias externas de la aplicación, lo que facilita decidir cuáles son las partes de los sistemas en desarrollo que son más susceptibles a un uso mal intencionado. Dicha solución actualmente cuenta con un conjunto de limitaciones en cuanto a la aplicación de nuevas técnicas para la implementación del análisis y gestión de riesgos de seguridad. Por tal motivo es conveniente la incorporación de nuevos métodos para el cálculo del riesgo que una amenaza supone sobre un determinado activo, nuevos artefactos de seguridad para tener un mayor control sobre los riesgos asociados a cada proyecto de análisis y las diferentes vías para contrarrestarlos. Dado que la herramienta no posee una forma de gestionar los riesgos de seguridad sería de utilidad la incorporación de técnicas diseñadas para este fin como la descripción de casos de mal uso y especificación de requisitos de seguridad, permitiendo tener una visión más clara de las acciones a acometer para mejorar la seguridad del proyecto en cuestión, por lo que es necesario desarrollar una nueva propuesta de solución.

2.3 Propuesta de la solución.

En la presente investigación se propone como solución a la problemática planteada el desarrollo de una segunda versión a la herramienta Segursoft. Esta herramienta permitirá al Centro TLM poner en marcha un nuevo conjunto de buenas prácticas desde las etapas iniciales del proceso de desarrollo de software mejorando así la gestión de la seguridad en los productos y servicios que brinda el Centro. Estas buenas prácticas se basan fundamentalmente en el análisis y gestión de riesgos de seguridad, para lo cual se mantendrán las prestaciones que anteriormente brindaba la herramienta como son la gestión de roles de usuario, de los principales componentes (tipos de elementos software que entran en la fabricación de las aplicaciones: archivos, paquetes, bibliotecas cargadas dinámicamente, etc.), de las dependencias externas (cualquier relación externa a dicho sistema), de los activos del proyecto y las amenazas asociadas a estos. La herramienta continuará brindando también una biblioteca con las amenazas más comunes de las aplicaciones, mantendrá la generación de informes y la modelación de diagramas de casos de abuso y árboles de ataque.

Para la nueva versión se le incorporará primeramente una base de datos para facilitar la gestión de la información que se procesa y darle así mayor seguridad a los datos que el usuario proporcione, se le adicionará además el uso de una nueva técnica para el cálculo del riesgo que suponen las amenazas, basada en la metodología MAGERIT. En el modelado de amenazas la herramienta brindará la posibilidad de modelar nuevos diagramas como son el diagrama de activos, el diagrama de casos de mal uso y el grafo de ataque. Finalmente se le incorporará la posibilidad de describir los casos de mal uso y especificar los requisitos de seguridad, luego el usuario podrá obtener un informe de estos en formato pdf facilitándole la gestión de la seguridad en los proyectos del Centro TLM.

2.4 Procesos del negocio.

A continuación se describen los procesos del negocio pertenecientes a la herramienta Segursoft en su versión 2.0. Para lograr una mayor claridad de los procesos mencionados se modelaron además diagramas de procesos del negocio utilizando la técnica IDEF0 mediante la herramienta Visio perteneciente al paquete Microsoft Office, de esta forma se facilita la comprensión de cada proceso con sus correspondientes entradas y salidas. En los diagramas referentes a las ilustraciones 7 y 8 se expone la secuencia de dichos procesos y se escriben de color rojo las nuevas funcionalidades que implementará la herramienta.

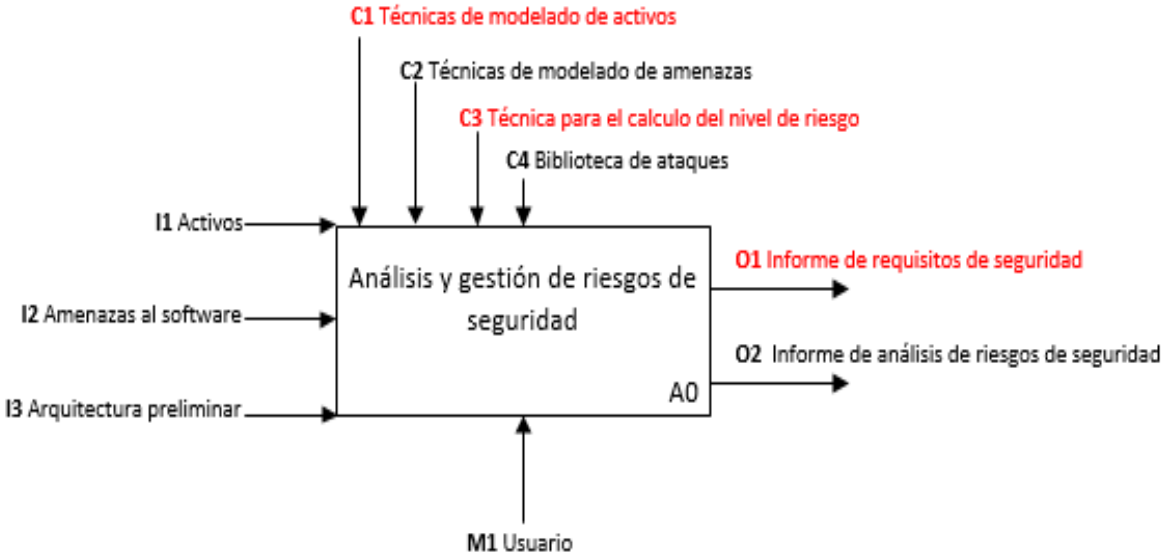


Ilustración 7 Diagrama de flujo del proceso análisis y gestión de riegos de seguridad. Fuente: Elaboración propia.

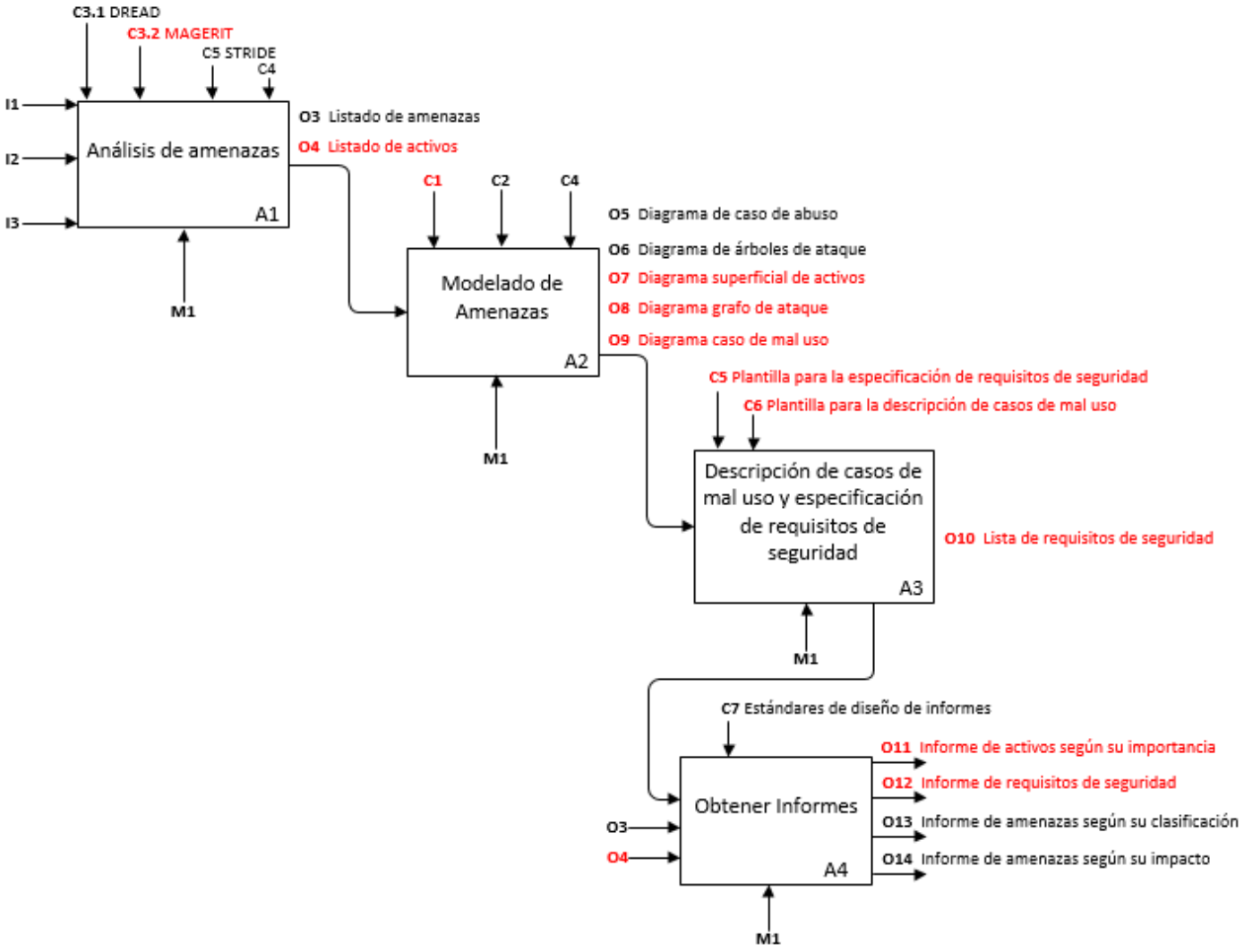


Ilustración 8 Diagrama de flujo para describir los subprocesos del sistema.
Fuente: Elaboración propia.

2.4.1 Análisis y gestión de riesgos.

El proceso base análisis y gestión de riesgos debe comenzar con el conocimiento previo de cuáles son los activos de la aplicación que se deben proteger y un estudio de la arquitectura preliminar del software, por lo que la aplicación primeramente permite al usuario identificar los activos, roles de usuarios, componentes y dependencias externas de la aplicación para luego realizar la inserción de las amenazas y asociarlas a los activos con los cuales se relacionan. Este proceso requiere total conocimiento de dichos datos por parte del usuario para obtener los resultados esperados ya que es este paso el que da inicio a todo el flujo de la

aplicación. Seguidamente el usuario podrá realizar los controles de seguridad y obtener los respectivos informes para realizar el análisis de las amenazas. Finalmente el sistema permite realizar la especificación de requisitos de seguridad dando la posibilidad de exportar un pdf con los datos correspondientes a estos.

2.4.2 Descripción de los subprocesos.

2.4.2.1 Análisis de amenazas.

Luego de ser especificados los activos más importantes, las amenazas que pueden afectar al software y la arquitectura preliminar de la aplicación, se debe proceder a clasificar y evaluar las amenazas que pueden afectar el sistema y los activos que este contiene. Dichas amenazas se clasifican según su tipo utilizando el método STRIDE, este debe su nombre a la traducción al inglés de los parámetros que utiliza para realizar la clasificación, los cuales son Suplantación de identidad, Manipulación de datos, Repudio, Revelación de información, Denegación de servicios y Elevación de privilegios. Luego las amenazas se priorizan de acuerdo a su nivel de riesgo utilizando el método DREAD o la metodología MAGERIT dependiendo de la preferencia del usuario. Finalmente se obtiene un listado con las amenazas clasificadas, priorizadas y con sus contramedidas asociadas y un listado con los activos pertenecientes al proyecto en análisis y sus respectivos niveles de importancia.

2.4.2.2 Modelado de amenazas.

Luego de haber identificado las posibles amenazas que pudieran afectar a la aplicación, el sistema permite al usuario diseñar los artefactos necesarios para realizar el modelado de las medidas de seguridad que serán ejecutadas para mitigar el impacto de dichas amenazas. Como entrada a este proceso se le proporciona la lista de amenazas identificadas y de los activos pertenecientes al proyecto y como resultado se obtiene un modelo de amenazas compuesto por los diagramas de árboles de ataque, grafos de ataque, casos de abuso, casos de mal uso y diagramas de activos.

2.4.2.3 Descripción de casos de mal uso y especificación de requisitos de seguridad.

Utilizando los diagramas de casos de mal uso y casos de abuso el usuario tiene la posibilidad de describir amenazas y especificar requisitos de seguridad a través de los estereotipos que representan los casos de mal uso y casos de uso de seguridad respectivamente. Como resultado se obtiene una lista almacenada en la base de datos conteniendo todos los requisitos de seguridad pertenecientes al proyecto en cuestión.

2.4.2.4 Obtener informes.

Luego de obtener el listado con las amenazas clasificadas y priorizadas, el modelo de amenazas y la lista de los requisitos de seguridad, se procede a generar los informes teniendo en cuenta el estándar de diseño definido y la información obtenida con la gestión de las amenazas según su impacto y su clasificación.

Luego de haber obtenido los informes el sistema brinda la posibilidad de obtener un listado en formato PDF de los requisitos de seguridad que requiere el software que se está analizando. Este subproceso constituye un paso clave para realizar la gestión de seguridad en un proyecto y obtener resultados satisfactorios. De esta forma se da fin al proceso de negocio análisis y gestión de riesgos de seguridad.

2.5 Exploración.

En esta fase se define el alcance real del proyecto, además de realizarse la entrevista con el(los) cliente(s) para definir las llamadas historias de usuario y con ellas lograr determinar los requisitos planteados por el cliente, es en esta fase también donde el equipo de desarrollo se familiariza con las tecnologías, herramientas y prácticas que utilizará en el proyecto que se desarrolla. La duración de esta fase oscila entre pocas semanas y pocos meses, dependiendo tanto de la experiencia como del conocimiento que tenga el equipo desarrollador.

2.5.1 Arquitectura del sistema.

La arquitectura de un sistema de software define al sistema en términos de componentes computacionales e interacciones entre esos componentes. El diseño de la arquitectura de un sistema, es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes conforman el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada. Debido a que la nueva versión del sistema incluirá una base de datos para guardar la información y eliminará los ficheros se hará especial énfasis en la arquitectura del nuevo software mostrando los cambios que se le van a realizar.

El patrón arquitectónico aplicado es el de n capas (Layers), pues el sistema está estructurado específicamente en 3 capas: presentación, negocio y acceso a datos, donde el objetivo principal es separar los diferentes aspectos del desarrollo, es decir, las cuestiones de presentación, lógica de negocio y acceso

a datos, permitiendo que un cambio en una de las capas no genere cambios en las demás. La arquitectura del sistema puede verse representada en la Ilustración 9, en la capa de presentación se encuentran las clases que tienen la responsabilidad de visualizar el contenido al usuario, la capa de negocio incluye las clases que implementan la lógica del negocio y la capa de acceso a datos contiene las clases que se relacionan con la base de datos permitiendo la manipulación de estos. La comunicación entre las capas se realiza a través de interfaces. (27)

Debido a que este fue el patrón utilizado en la primera versión se decide utilizarlo en la solución propuesta, por las facilidades que ofrece esta arquitectura será modificada la capa de acceso a datos pero no serán afectadas radicalmente las demás capas.

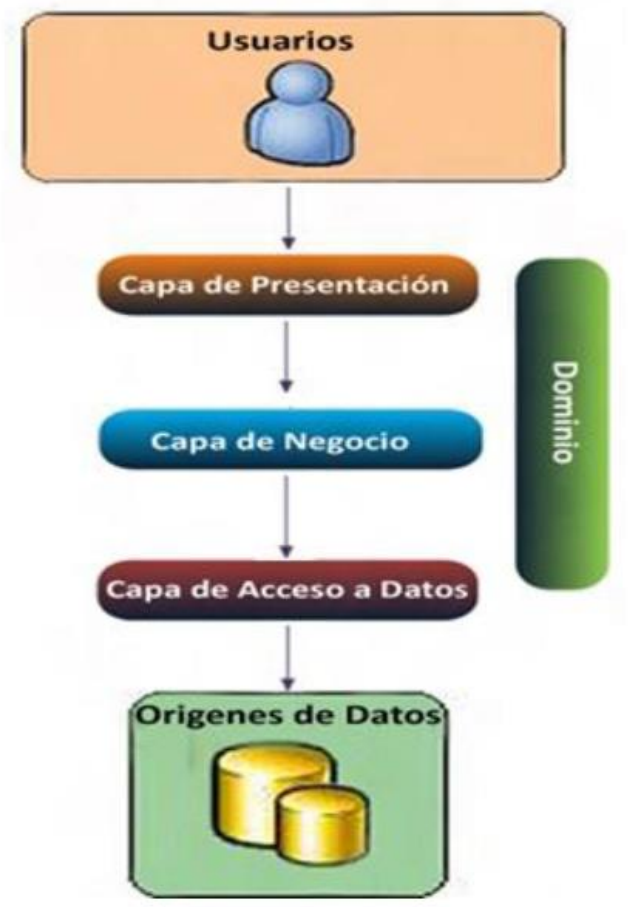


Ilustración 9 Arquitectura del sistema
Fuente: (27)

Capa de presentación.

En la capa de presentación se encuentran las interfaces que interactúan con el usuario con sus respectivas controladoras, además de los ficheros de configuración de Spring. Esta contiene los formularios y paneles que se encargan de visualizar el contenido al usuario, también posee las clases controladoras de Interfaz de Usuario que tienen la responsabilidad de procesar la información proporcionada por el usuario y de establecer la comunicación con la capa de negocio.

Capa de negocio.

La capa de negocio contiene las clases de interfaz con las clases que las implementan, unido al fichero de configuración de Spring. Esta contiene las clases que especifican qué se debe hacer pero no su implementación, además de las clases que implementan la lógica del negocio, estas clases se encargan de la comunicación con la capa de acceso a datos a través de la interfaz IADProyecto.

Capa de acceso a datos.

La capa de acceso a datos contiene la clase DAOProyectosImpl que se encarga de manipular las acciones de un proyecto en la base de datos, permitiendo el acceso y la modificación de los datos de un proyecto, unido al fichero de configuración de Spring. Debido a que a la nueva versión de la herramienta se le implementará una base de datos y se eliminará la anterior forma de guardar los datos en ficheros la capa de acceso a datos será modificada. Las ilustraciones 10 y 11 muestran la estructura de la capa **accesodato** y el modelo entidad relación respectivamente para un mayor entendimiento de los cambios realizados sobre esta capa.

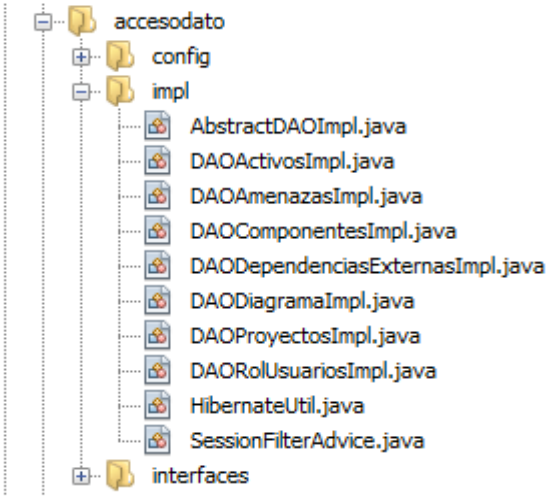


Ilustración 10 Estructura de la capa accesodato.
Fuente: Elaboración propia

- **Paquete impl:** En la nueva versión, como se muestra en la ilustración 7 se le agregarán a este paquete un conjunto de clases. La clase AbstractDAOImpl tiene la responsabilidad de realizar las acciones directas con la base de datos (métodos en común de los elementos gestionados) y las clases DAO restantes contendrán solo los métodos específicos asociados a cada elemento a persistir en la base de datos.
- **Paquete interfaces**: Anteriormente contenía la clase interface IADProyecto que especificaba qué se debía hacer pero no contenía su implementación. En la nueva versión se le incorporarán las interfaces de las clases contenidas en el paquete impl usando la misma estructura de la clase IADProyecto.
- **Paquete config:** tiene los ficheros de configuración de Spring que se encargan de guardar el contexto donde se encuentran configuradas las clases que serán inyectadas a través de Spring, específicamente las clases de acceso a datos. Además contiene un archivo de propiedad asociado a la configuración de hibernate.

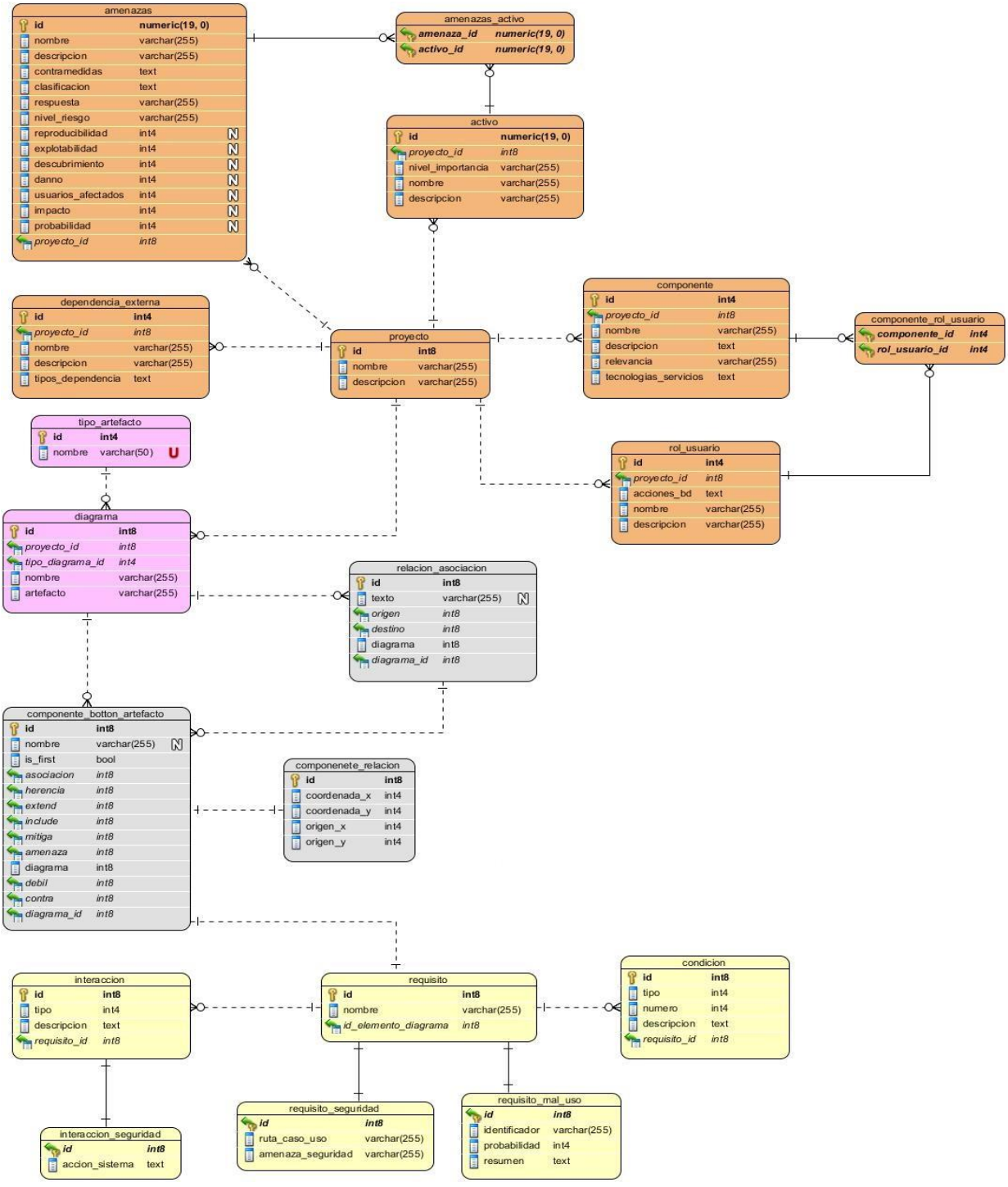


Ilustración 11 Modelo entidad relación.
Fuente: Elaboración propia.

2.5.2 Funcionalidades del sistema.

Las funcionalidades del sistema no son más que los requerimientos que establece el cliente para satisfacer sus necesidades con respecto al producto que solicita. Seguidamente se listan las funcionalidades para la nueva versión de Segursoft.

- Cambiar el método de almacenamiento de los datos, de ficheros a base de datos.
- Acceder a un proyecto de análisis.
- Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT.
- Diseñar diagrama superficial de activos.
- Diseñar diagrama de mal uso.
- Diseñar grafo de ataque.
- Generar reporte de activos según su importancia.
- Especificar requisitos de casos de uso de seguridad.
- Especificar descripción de casos de mal uso.
- Generar reporte de requisitos de seguridad.

2.5.3 Historias de usuario.

Las HU, constituyen la técnica definida por XP para mostrar los requerimientos de software. En ellas se describen brevemente las características con las que el sistema debe contar, requisitos que deben ser planteados por el cliente y deben ser escritas por el mismo. Para su redacción se debe usar un lenguaje sencillo permitiendo una fácil comprensión de estas por los programadores para asegurar una rápida implementación. Cuando llegue el momento de la implementación, los desarrolladores podrán dialogar directamente con el cliente para obtener todos los detalles necesarios.

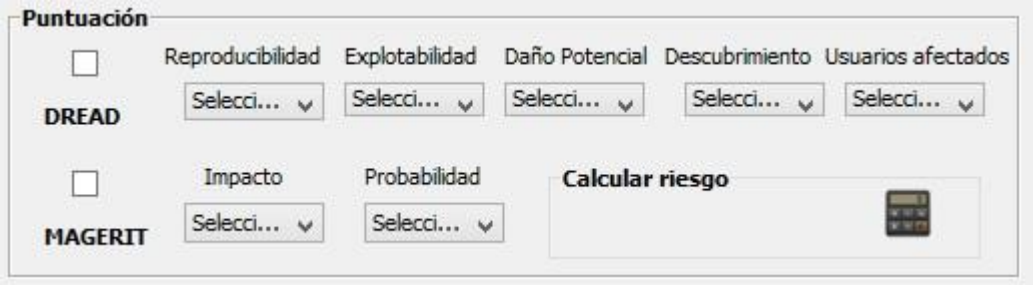
2.5.3.1 Usuarios del sistema.

Como usuarios del sistema se definen al analista, el arquitecto y líder del proyecto debido a que son estos los que tienen una relación más directa con la gestión de la seguridad en un software en desarrollo. Para estandarizar la descripción de las HU se utilizará el rol analista para especificar el usuario que hará uso de la aplicación.

Seguidamente se presenta una de las HU definidas por el cliente, el resto de estas podrán encontrarse en los ANEXOS.

Historia de usuario Nro.3: Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT.

Tabla 2 Historia de usuario #3

Historia de usuario	
Número: 3	**Usuario:** Analista.
Nombre de Historia de Usuario: Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT.	
Prioridad en negocio: Media.	**Riesgo en Desarrollo:** Medio.
Puntos estimados: 1	**Iteración Asignada:** 2
Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema permite al usuario hacer uso de la metodología MAGERIT, la misma posibilitará evaluar el nivel de riesgo que supone una amenaza haciendo uso de los criterios de impacto y probabilidad que tiene dicha amenaza sobre un determinado activo.	
Observaciones:	
Prototipo de interfaz: 	

2.5.4 Características no funcionales del sistema.

Las características no funcionales en un software son muy importantes ya que estas están compuestas por las cualidades que todo sistema debe portar para prestar el servicio para el que fue concebido. En la solución propuesta se han identificado las siguientes características no funcionales:

Usabilidad.

- Se necesitará una preparación previa para operar con la herramienta ya que la usarán analistas que pueden no tener conocimientos suficientes de la seguridad informática, esta no debe ser muy extensa porque solo se debe instruir al usuario en el flujo de las acciones que debe seguir para realizar el análisis y gestión de riesgos de seguridad y cómo la herramienta facilita esta buena práctica.

Portabilidad.

- Para que la herramienta pueda ser ejecutada se necesita tener instalada la máquina virtual de Java.
- Para el uso solo se bebe copiar el instalador y realizar la instalación de esta.

Hardware.

- Para la instalación de la herramienta es necesario disponer de una computadora de 512 MB de RAM o superior capaz de responder en el menor tiempo posible a las necesidades del usuario. Debe usarse una computadora con una capacidad de almacenamiento mínima de 40 GB para almacenar la información que gestiona la herramienta.

Software.

- La herramienta puede ser usada tanto en sistemas operativos Windows como Linux y se debe tener instalada la máquina virtual de Java.

Apariencia o interfaz externa.

- Debe ofrecer una interfaz amigable, fácil de operar y en la cual el usuario se sienta cómodo al trabajar sobre esta.
- La herramienta mantendrá la línea de diseño establecida para ella, misma tipología de botones, mismos colores de las ventanas, misma letra etc.

2.6 Planificación.

La metodología XP para el desarrollo de software establece que en la fase de planificación se establecen las prioridades para los requerimientos recogidos (HU definidas). Después de haberlas recopilado los programadores evalúan rápidamente el tiempo de desarrollo de cada una, luego para su implementación le son asignadas a cada una un conjunto de tareas de ingeniería las cuales detallan cada paso a seguir para que sean realizadas tal como las solicita el cliente, finalmente se establecen iteraciones de acuerdo con las prioridades asignadas a cada requerimiento. El tiempo de cumplimento de cada iteración se obtiene al sumar el tiempo de desarrollo de las HU que están contenidas dentro de ella. Una vez realizadas estas estimaciones, se organiza una reunión de planificación con los diversos actores del proyecto (cliente, desarrolladores, gerentes, etc.) para de conjunto establecer un plan o cronograma de entregas.

2.6.1 Estimación del esfuerzo por HU.

XP no tiene métricas predeterminadas para medir el desempeño del proyecto en cuestión, lo cual permite utilizar cualquier criterio para medir el alcance del proyecto. En este caso se utilizará la métrica que toma como medida el punto, al que se considerará como una semana ideal de trabajo en la cual se trabaja el tiempo planeado sin interrupción alguna. En los casos donde se requiera solo un número de días inferior a una semana de trabajo se utilizara la fracción x/5 donde x representa el número de días necesarios y el 5 el número de días hábiles de trabajo en una semana. Esto se realiza para reducir al mayor grado posible el tiempo de desarrollo del proyecto.

Tabla 3 Estimación de esfuerzo por historias de usuarios.

Historias de usuario	Puntos estimados
1. Cambiar método de almacenamiento de los datos, de ficheros a base de datos.	2
2. Acceder a un proyecto de análisis.	1
3. Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT.	1
4. Diseñar diagrama superficial de activos.	2

5. Diseñar diagrama de mal uso.	2
6. Diseñar grafo de ataque.	2
7. Generar reporte de activos según su importancia.	1
8. Especificar requisitos de casos de uso de seguridad.	1
9. Especificar descripción de casos de mal uso.	1
10. Generar reporte de requisitos de seguridad en formato pdf.	1

2.6.2 Plan de iteraciones.

Después de ser identificadas las HU y estimado el esfuerzo necesario para la realización de cada una de ellas se procede a la división de su implementación en cuatro iteraciones para lograr presentar de forma satisfactoria las entregas periódicas al cliente que establece la metodología XP. Se especifican también las tareas de ingeniería pertenecientes a cada iteración:

Iteración 1.

La siguiente iteración contiene las HU número 1 y 2 de las cuales la número 2 es crítica para el sistema debido a que el resto de las UH dependen de esta para su implementación y se hace imprescindible implementarla antes de pasar al desarrollo de las próximas HU, la número 2 presenta un alto valor para el negocio ya que para el usuario poder realizar el resto de las acciones en el sistema es necesario primero acceder a un proyecto de análisis. En esta iteración se desarrollarán las tareas de ingeniería número 1, 2, 3 y 4 pertenecientes a las HU antes mencionadas.

Iteración 2.

Esta será la encargada de la implementación de las HU 3 y 4 ya que solo luego de que sean clasificadas todas las amenazas estas podrán ser modeladas y priorizadas para facilitar su análisis y gestión. El modelado de activos y sus relaciones mediante el diagrama superficial de activos es un paso clave en el análisis de amenazas, es por tal motivo que se seleccionaron estas HU para ser desarrolladas con mayor prioridad. En esta iteración se desarrollarán las tareas de ingeniería número 5, 6, 7 y 8.

Iteración 3.

La siguiente iteración será la encargada de la implementación de las HU 5 y 6 las cuales permiten la modelación de los diagramas de mal uso y el grafo de ataque, estos diagramas son necesarios en el análisis de amenazas debido a que muestran las diferentes formas en que puede ser atacado un sistema y las vías a usar para efectuar dichos ataques, esta información luego de ser analizada es necesaria para lograr llevar a cabo satisfactoriamente las técnicas de gestión de riesgos. En esta iteración se desarrollarán las tareas de ingeniería número 9, 10, 11, 12, 13 y 14.

Iteración 4.

En la iteración 4 se desarrollarán las HU 7, 8, 9 y 10 las cuales se refieren a la descripción de los casos de mal uso, la elicitación de requisitos de seguridad y finalmente la generación de los reportes de activos y de requisitos de seguridad, dando fin al proceso de análisis y gestión de riesgos de seguridad en la herramienta Segursoft. En esta iteración se desarrollarán las tareas de ingeniería número 15, 16, 17, 18 y 19.

2.6.3 Plan de duración de iteraciones.

El plan de duración de las iteraciones es, como su nombre lo indica, el encargado de mostrar la duración de cada una de las iteraciones. En él se muestran las iteraciones que fueron planificadas para el desarrollo de la solución propuesta, dentro de cada una se muestran las HU que les fueron asignadas y a continuación se especifica el número total (en semanas) de duración de cada iteración, dada por la suma de la duración de las HU que las componen.

Tabla 4 Plan de iteraciones

Iteración	Orden de la historias de usuario a implementar.	Duración total
1	Cambiar método de almacenamiento de los datos, de ficheros a base de datos. Acceder a un proyecto de análisis.	3 semanas
2	Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT. Diseñar diagrama superficial de activos.	3 semanas

3	Diseñar diagrama de mal uso. Diseñar grafo de ataque.	4 semanas
4	Generar reporte de activos según su importancia. Especificar requisitos de casos de uso de seguridad. Especificar descripción de casos de mal uso. Generar reporte de requisitos de seguridad en formato pdf.	4 semanas

2.6.4 Plan de entregas.

A continuación se presenta el plan de entregas de las iteraciones para llevar a cabo el desarrollo de la versión propuesta a la herramienta Segursoft, en él se representa una gráfica del tiempo detallando las fechas de inicio de cada iteración y el producto que debe presentarse al cliente en dicha fecha.

Tabla 5 Plan de entregas.

Iteraciones\Fecha	**6 de marzo del 2015**	**27 de marzo del 2015**	**17 de abril del 2015**	**15 de mayo del 2015**	**12 de junio del 2015**
Iteración 1	Comenzada	Terminada	Terminada	Terminada	Terminada
Iteración 2	No comenzada	Comenzada	Terminada	Terminada	Terminada
Iteración 3	No comenzada	No comenzada	Comenzada	Terminada	Terminada
Iteración 4	No comenzada	No comenzada	No comenzada	Comenzada	Terminada
Producto	-	**Segursoft** V1.1	**Segursoft** V1.2	**Segursoft** V1.3	**Segursoft** V2.0

2.7 Conclusiones parciales.

El desarrollo de este capítulo ha permitido comprender la estructura y dinámica del sistema al cual se le desarrollará una segunda versión. Como resultado, se han obtenido los requerimientos del sistema a través de las HU, planificándose en un conjunto de iteraciones. Se redefinió la arquitectura base de la aplicación

para cumplir con los requerimientos solicitados por el cliente. A partir de los resultados obtenidos en la planificación y análisis del sistema se puede proceder entonces al diseño, implementación y despliegue de la nueva versión de la herramienta Segursoft, etapas finales en el proceso de mantenimiento de la misma, las cuales se presentan en el próximo capítulo.

CAPÍTULO 3: DISEÑO, IMPLEMENTACIÓN Y PRUEBAS

3.1 Introducción.

La codificación es un proceso que se realiza en forma paralela con el diseño y la cual está sujeta a varias observaciones por parte de XP. Primeramente esta metodología plantea que el cliente debe estar siempre presente para solucionar las dudas que puedan surgir en el grupo de desarrollo debido a que las HU están escritas en un lenguaje demasiado simple como para describir completamente cada detalle del sistema que se crea. XP también plantea que se debe trabajar en pareja a la hora de codificar el software ya que cuando se trabaja en dúo, lejos de retrasar la codificación, se obtiene un diseño de mejor calidad y un código más organizado y con menores errores que si trabajasen separados. Así como se recomienda que la programación se haga siempre en parejas ubicadas en una única computadora, también se aconseja que estas se vayan rotando no solo de compañero sino de partes del proyecto a implementar, con el fin de que se logre tener una propiedad colectiva del código. En el presente capítulo se expondrán las principales características referentes a las etapas de diseño, codificación y pruebas de la solución propuesta. Se especifican además los patrones de diseño a utilizar, los cuales dan solución a diversos problemas y ayudan a cumplir varios principios o reglas de diseño y se definirán las tarjetas CRC para establecer las relaciones de colaboración entre cada clase implementada. Finalmente se describen las tareas de la ingeniería para profundizar más en cada una de las funcionalidades especificadas por el cliente y se definen los casos de pruebas asociados a cada una de las HU para validar el buen funcionamiento de la solución a implementar.

3.2 Diseño.

Para el diseño del software que se desarrolla sobre la metodología XP se requiere ante todo un mínimo nivel de complejidad. Pero un diseño claro y simple no significa que deba ser falto de información o pobre de explicación, es por eso que en esta fase se describen detalladamente las tarjetas CRC (Clase Responsabilidad Colaboración) las cuales reflejan cuales son las clases que tienen las mayores responsabilidades y sus relaciones con otras clases, también se mencionan y explican los patrones de diseño utilizados.

3.2.1 Tarjetas CRC.

Las tarjetas CRC son un inventario de las clases que se necesitan para implementar el sistema y la forma en que van a interactuar, permiten también que el equipo completo contribuya en la tarea del diseño. Estas registran el nombre de las clases, sus responsabilidades y las otras clases con las que colaboran. (28)

Seguidamente se presentan 4 de las tarjetas CRC definidas para el desarrollo de la solución, las cuales hacen referencia al proceso de gestión de un activo, el resto de estas podrán encontrarse en los ANEXOS.

Tabla 6 CRC 1 ActivosControladora.

Clase ActivosControladora	
Responsabilidad	**Colaboración**
Se encarga de recoger los datos referentes a un activo proporcionados por el formulario FrmActivos. Luego de haber recogido estos datos, la clase ActivosControladora proporciona dicha información a la clase IGestionarActivo.	FrmActivos IGestionarActivo

Tabla 7 CRC 2 IGestionarActivo.

Clase IGestionarActivo	
Responsabilidad	**Colaboración**
Esta clase hace referencia a los métodos de la clase GestionarActivoImpl la cual se encarga de recoger los datos referentes a un activo proporcionados por la clase ActivosControladora y se los suministra a la clase DAOActivosImpl.	ActivosControladora GestionarActivoImpl DAOActivosImpl

Tabla 8 CRC 3 DAOActivosImpl.

Clase DAOActivosImpl

Responsabilidad	Colaboración
Esta clase extiende de AbstractDAOImpl complementando los métodos de esta, además se encarga de recoger los datos referentes a un activo proporcionados por la clase GestionarActivoImpl y los guarda en la base de datos.	GestionarActivoImpl

Tabla 9 CRC 3 AbstractDAOImpl.

Clase AbstractDAOImpl	
Responsabilidad	**Colaboración**
Esta clase contiene todos los métodos comunes a los elementos a persistir en el sistema como Guardar, Eliminar, etc. Se encarga de recoger los datos referentes a una determinada clase y los guarda en la base de datos.	GestionarActivoImpl GestionarAmenazaImpl GestionarDependenciaImpl GestionarDiagramasImpl GestionarProyectoImpl GestionarComponenteImpl GestionarRolUsuarioImpl

3.2.2 Patrones de diseño.

Un patrón de diseño es una solución a un problema de diseño. Se consideran como procedimientos para solucionar diversos problemas del mismo tipo y son la base para la búsqueda de soluciones a dificultades comunes en el desarrollo de software. A continuación se describen los patrones utilizados en la implementación de la nueva solución (27):

Patrón Observer.

Problema: Se necesita lograr que las clases observadoras tengan la información de cuándo se produce un cambio de estado en el objeto observado.

Solución: Los objetos observadores se añaden a una lista, y el objeto observado notificará a todos los objetos de esta lista cuando se produzca el cambio. El observador que tenía la tarea de estar pendiente del cambio, ahora está a la espera de que el observado le avise. (27)

Implementación: En la aplicación el objeto observado es un proyecto, mientras los objetos observadores son los activos, roles de usuario, componentes, dependencias externas, amenazas y artefactos. De esta forma cuando se realiza alguna modificación (adicionar, modificar, eliminar) en alguno de los objetos observadores, se tiene la información de en qué proyecto se está trabajando, y en caso de que ocurra algún cambio la clase Observado, la cual contiene un objeto proyecto, notifica inmediatamente a los objetos observadores.

Patrón Inyección de Dependencia.

Problema: Se necesitan crear objetos y componentes reutilizables.

Solución: La inyección de dependencias garantiza la inyección a cada objeto de los objetos necesarios según las relaciones plasmadas en un fichero de configuración.

Implementación: Se delega en Spring la creación de los objetos.

Patrones GRASP.

Los patrones GRASP (General Responsability Assigment Software Patterns) constituyen patrones de diseño de software para asignación de responsabilidades a objetos. Estos son considerados como una serie de buenas prácticas recomendables en el diseño de software ya que ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas a los cuales ellos dan respuesta. A continuación se enuncian los patrones utilizados en el diseño de la solución:

Experto: Constituye el principio básico de asignación de responsabilidades. Este patrón plantea la pregunta:

- ¿Cuál es el principio fundamental en virtud del cual se asignará las responsabilidades a los objetos?

A la que establece la respuesta de asignar una responsabilidad a la clase que tiene la información necesaria para cumplirla. Es por esto que fue seleccionado este patrón para la asignación de responsabilidades a las clases, ejemplo de ello lo constituye la clase GestionarActivoImpl la cual se encarga de adicionar, eliminar y modificar un activo y cuenta con la información necesaria para cumplir con esas responsabilidades como se muestra en la Ilustración 12.

```
41      public GestionarActivoImpl() {...}
44
45      @Override
        public Activo adicionarActivo(String nombreaActivo, String paranoia, String descripcion) throws Exception {...}
64
65      @Override
        public boolean validarObservado() {...}
69
70      @Override
        public Activo eliminarActivo(Activo activo) {...}
75
76      @Override
        public Activo modificarActivo(String nombreActivo, String descripcion, String paranoia, String nombreViejo) {..
88
89      @Override
        public boolean validarSiExiste(String nombre) {...}
100
101     @Override
        public List<Activo> listarActivos() {...}
105
106     @Override
        public Activo obtenerActivoPorNombre(String elementAt) {...}
110
111     @Override
        public Activo buscarActivoPorId(Long id) {...}
115
```

Ilustración 12 Patrón experto
Fuente: Elaboración propia

Alta Cohesión.

Problema: Se necesita lograr que las clases trabajen en su misma área de aplicación.

Solución: Este patrón es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Caracteriza a las clases que están estrechamente relacionadas y consiste en colaborar con otros objetos para compartir el esfuerzo si la tarea a realizar es grande.

Implementación: A las clases controladoras de la capa de presentación y las clases de implementación de la capa negocio se le asignan responsabilidades con el objetivo de que trabajen en la misma área de aplicación y que no tengan mucha complejidad, por ejemplo para adicionar un componente la tarea se divide entre varias clases como se muestran en las Ilustraciones 13,14 y 15, la clase ActivosControladora se encarga de los cambios que ocurren en la presentación, la clase GestionarActivoImpl gestiona los cambios en el negocio y la clase DAOActivoImpl interactúa en la capa de acceso de datos, para finalizar la tarea de adicionar un componente.

```
23    public class ActivosControladora extends PanelControladora {
24
25        private FrmActivos panelActivo;
26        private IGestionarActivo gestionarActivo;
27        private ActivoPrincipalControladora activoprincipal;
28
29        public ActivoPrincipalControladora getActivoprincipal() {...}
32
33        public void setActivoprincipal(ActivoPrincipalControladora activoprincipal) {...}
36
37        public ActivosControladora() {...}
40
41        public IGestionarActivo getGestionarActivo() {...}
44
45        public void setGestionarActivo(IGestionarActivo gestionarActivo) {...}
48
49        public FrmActivos getFrmActivo() {...}
52
53        public void setPanelActivo(FrmActivos panelActivo) {...}
57
58        @Override
          public void mostrarFormulario() {...}
66
67        @Override
          public void inicializar() {...}
74
75        private final ActionListener agregarActivos = new ActionListener() {...};
116
117       private final ActionListener cancelar = new ActionListener() {...};
```

Ilustración 13 Clase ActivosControladora
Fuente: Elaboración propia

```
20    public class DAOActivosImpl  extends AbstractDAOImpl<Activo, Long> implements IDAOActivos {
21
22        public DAOActivosImpl() {...}
25
26        @Override
          public void guardar(String archivo, Object obj) throws IOException {...}
29
30        @Override
          public void cargar(String archivo) throws IOException, ClassNotFoundException {...}
33
34        @Override
          public List<Activo> getAllByProject(Proyecto p) {...}
45
46        @Override
          public Activo obtenActivoNombre(String nombre) throws HibernateException {...}
58
59        @Override
          public Boolean existeActivoEnProyecto(String nombreActivo, Long idProyceto) throws HibernateException {...}
72    }
73
```

Ilustración 14 Clase DAOActivosImpl
Fuente: Elaboración propia

```
41      public GestionarActivoImpl() {...}
44
45      @Override
        public Activo adicionarActivo(String nombreaActivo, String paranoia, String descripcion) throws Exception {
64
65      @Override
        public boolean validarObservado() {...}
69
70      @Override
        public Activo eliminarActivo(Activo activo) {...}
75
76      @Override
        public Activo modificarActivo(String nombreActivo, String descripcion, String paranoia, String nombreViejo)
88
89      @Override
        public boolean validarSiExiste(String nombre) {...}
100
101     @Override
        public List<Activo> listarActivos() {...}
105
106     @Override
        public Activo obtenerActivoPorNombre(String elementAt) {...}
110
111     @Override
        public Activo buscarActivoPorId(Long id) {...}
115 }
```

Ilustración 15 Clase GestionarActivoImpl
Fuente: Elaboración propia

Bajo Acoplamiento.

Problema: Se necesita lograr una escasa dependencia entre clases.

Solución: Realizar un diseño de clases independientes que puedan soportar los cambios de una manera fácil y permitan la reutilización. Acoplamiento bajo significa que una clase no depende de muchas clases.

Implementación: Al aplicar el patrón arquitectónico n capas se asegura el bajo acoplamiento, de forma tal que las cuestiones de presentación, negocio y acceso a datos están totalmente desligadas, propiciando que un cambio en una capa no afecte a las demás capas inferiores, esto se logra estableciendo la comunicación entre las capas mediante interfaces, por ejemplo la ModificarActivoAccionControladora de la capa de presentación se comunica a través de la interfaz IGestionarActivo con la capa de negocio, y la clase GestionarActivoImpl que implementa dicha interfaz se comunica con la capa de acceso a datos a través de la interfaz IDAOActivosImpl. En las Ilustraciones 16 y 17 se muestra un ejemplo de estas clases.

```
18  /**...*/
22  public class ModificarActivoAccionControladora extends PanelControladora {
23
24      private FrmActivos panelactivo;
25      private IGestionarActivo gestionarActivo;
26      private ActivoPrincipalControladora activoprincipal;
27
28      public ActivoPrincipalControladora getActivoprincipal() {...}
31
32      public void setActivoprincipal(ActivoPrincipalControladora activoprincipal) {...}
35
36      public FrmActivos getPanelactivo() {...}
39
40      public void setPanelactivo(FrmActivos panelactivo) {...}
43
44      public IGestionarActivo getGestionarActivo() {...}
47
48      public void setGestionarActivo(IGestionarActivo gestionarActivo) {...}
51
52      public ModificarActivoAccionControladora() {...}
54
55      @Override
        public void inicializar() {...}
84
85      private final ActionListener cancelar = new ActionListener() {...};
92
93      public String devolverRadio(Activo activo) {...}
```

Ilustración 16 Clase ModificarActivoAccionControladora
Fuente: Elaboración propia

```
41      public GestionarActivoImpl() {...}
44
45      @Override
        public Activo adicionarActivo(String nombreaActivo, String paranoia, String descripcion) throws Exception {
64
65      @Override
        public boolean validarObservado() {...}
69
70      @Override
        public Activo eliminarActivo(Activo activo) {...}
75
76      @Override
        public Activo modificarActivo(String nombreActivo, String descripcion, String paranoia, String nombreViejo)
88
89      @Override
        public boolean validarSiExiste(String nombre) {...}
100
101     @Override
        public List<Activo> listarActivos() {...}
105
106     @Override
        public Activo obtenerActivoPorNombre(String elementAt) {...}
110
111     @Override
        public Activo buscarActivoPorId(Long id) {...}
115 }
```

Ilustración 17 Clase GestionarActivoImpl
Fuente: Elaboración propia

3.3 Tareas de ingeniería.

Las tareas de ingeniería no son más que el desglose al que es sometida una historia de usuario para aumentar así su nivel de detalle y hacer más simple su comprensión e implementación por parte de los desarrolladores. A continuación se describe una de las tareas de ingeniaría pertenecientes a la solución propuesta, el resto de estas podrán encontrarse en los ANEXOS.

Tabla 10 Tarea de ingeniería #1.

Tarea de ingeniería.	
Número Tarea: 1	**Número Historia de Usuario: 1**
Nombre Tarea: Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados: 1**
Programador responsable: Addiel Velazco Padilla, Yordanis Calvo Rodríguez	
Descripción: Para la estimación del riesgo que supone una amenaza sobre un activo el sistema debe permitir el uso de la metodología MAGERIT. Dado que ya el sistema brinda también la posibilidad de estimar el riesgo mediante la técnica DREAD, se le debe pedir al usuario elegir una de las dos formas de hacerlo. Por tal motivo la nueva técnica debe ser incorporada junto a la ya existente en el formulario de adicionar amenaza para que el usuario seleccione la deseada. Una vez seleccionada la metodología MAGERIT el usuario debe evaluar el nivel de impacto (alto, medio y bajo) y probabilidad (alto, medio y bajo) de dicha amenaza. El sistema con estos valores debe realizar el cálculo del nivel de riesgo, el cual tendrá como salida los valores muy bajo, medio, alto y muy alto.	

3.4 Estándares de codificación.

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, este debe tender siempre a lo

práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutina. (29)

Para asegurar la legibilidad del código entre los desarrolladores y facilitar el mantenimiento y la actualización de la herramienta se utilizaron los estándares establecidos en la primera versión de la herramienta Segursoft, a continuación se describen cada uno de ellos (27).

Estándar 1: las interfaces de usuario (formularios y paneles) que se encuentran en el paquete formularios de la capa de presentación se nombran comenzando siempre con las siglas Frm.

Estándar 2: las clases ubicadas en el paquete controladoras de la capa de presentación se nombran comenzando siempre con mayúscula y terminando con la palabra Controladora.

Estándar 3: los métodos implementados deben comenzar siempre con minúscula y con un nombre sugerente a la funcionalidad que realizan.

Estándar 4: las clases interface de la capa de negocio se deben nombrar con mayúscula comenzando con la letra I seguido del nombre de la funcionalidad. Las clases interface de la capa de acceso a datos deben comenzar con las siglas IAD seguido del nombre de la funcionalidad.

Estándar 5: las clases que están ubicadas en el paquete impl de la capa de negocio deben tener el mismo nombre de la interface que implementan omitiendo la letra I y finalizando con las siglas Impl. Las clases de la capa de acceso a datos que implementan las interface de esa misma capa deben nombrarse de igual forma omitiendo la letra I seguido de las siglas AD y el nombre de la funcionalidad.

3.5 Pruebas.

El proceso de pruebas es un elemento primordial y uno de los pilares fundamentales de la metodología XP, el cual estimula a los desarrolladores a probar constantemente el software tanto como sea posible. Mediante esta filosofía se reduce el número de errores no detectados así como el tiempo entre la introducción de este

en el sistema y su detección. Dicha metodología divide las pruebas en varios grupos entre los que se encentran (30):

Pruebas de Aceptación(o pruebas funcionales): Son las pruebas destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente.

Pruebas Unitarias: Son las pruebas implementadas por los desarrolladores, encargadas de verificar el código.

3.5.1 Pruebas de Aceptación.

Las pruebas de aceptación son pruebas de caja negra definidas por el cliente para cada historia de usuario, y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. En efecto, las pruebas de aceptación corresponden a una especie de documento de requerimientos en XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención. (30)

Estas pruebas deben su importancia, en especial, a que miden el nivel de satisfacción del cliente con cada iteración concluida, además de que marcan el final de esta y el comienzo de la próxima. Por lo que a continuación se muestran una serie de casos de prueba, los cuales servirán como muestra visual del proceso de pruebas realizadas a la aplicación propuesta.

Dichos casos de pruebas se describirán en tablas que contendrán los siguientes campos:

- **Clases Válidas**: Se hará la descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tendrá en cuenta cada una de las entradas válidas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado.
- **Clases Inválidas:** Se hará la descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tendrá en cuenta cada una de las posibles entradas inválidas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado y cómo responde el sistema.
- **Resultado Esperado**: Se hará una breve descripción del resultado que se espera ya sea para entradas válidas o entradas inválidas.

- **Resultado de la Prueba**: Se hará una breve descripción del resultado que se obtiene.
- **Observaciones:** Algún señalamiento o advertencia que sea necesario hacerle a la sección que se está probando.

La evaluación de la prueba realizada se hará según el resultado de la misma, la tendrá uno de los tres resultados que a continuación se describen:

- **Satisfactorio:** Cuando el resultado de la prueba es exactamente el esperado por el usuario.
- **No Satisfactorio:** Cuando el resultado de la prueba realizada genera un error de codificación en la aplicación o muestra como resultado elementos no deseados o fuera de contexto, trayendo como consecuencia que la funcionalidad requerida por el cliente no tenga resultado, lo que invalida también la UH.

Prueba de aceptación #1: Historia de usuario #3 (Evaluar nivel de riesgo de una amenaza mediante la metodología MAGERIT).

✓ Prueba: El usuario selecciona la metodología MAGERIT y valora el impacto y probabilidad de la amenaza.

Tabla 11 Prueba de aceptación # 1.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Alto y la probabilidad en Alta y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Muy Alto	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Alto y la		El sistema muestra el nivel de riesgo calculado con valor Alto	Satisfactorio.	En el sistema debe haber un activo

probabilidad en Medio y calcula el nivel de riesgo				previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Alto y la probabilidad en Bajo y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Alto	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Medio y la probabilidad en Alto y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Alto	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Medio y la probabilidad en Medio y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Medio	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Medio y la probabilidad en Bajo y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Medio	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Bajo y la		El sistema muestra el nivel de riesgo calculado con valor Medio	Satisfactorio.	En el sistema debe haber un activo

probabilidad en Alto y calcula el nivel de riesgo				previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Bajo y la probabilidad en Medio y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Bajo	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Bajo y la probabilidad en Bajo y calcula el nivel de riesgo		El sistema muestra el nivel de riesgo calculado con valor Bajo	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
	El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Seleccione y la probabilidad en Seleccione y calcula el nivel de riesgo	El sistema le muestra un mensaje informativo con el siguiente texto “Debe dar un valor a los parámetros de puntuación”	Satisfactorio.	En el sistema debe haber un activo previamente almacenado.
	El usuario selecciona la metodología MAGERIT y evalúa el impacto	El sistema le muestra un mensaje informativo para indicar que debe dar valores a los parámetros de puntuación.	No satisfactorio	En el sistema debe haber un activo previamente almacenado.

	del riesgo en Seleccione y la probabilidad en Alto, Medio o Bajo y calcula el nivel de riesgo			
	El usuario selecciona la metodología MAGERIT y evalúa el impacto del riesgo en Alto, Medio o Bajo y la probabilidad en Seleccione y calcula el nivel de riesgo	El sistema le muestra un mensaje informativo para indicar que debe dar valores a los parámetros de puntuación.	No satisfactorio	En el sistema debe haber un activo previamente almacenado.

Conclusiones de las pruebas de aceptación.

Para la aplicación de las pruebas de aceptación como se muestra en la Ilustración 18 se aplicaron en la primera iteración un total de 25 pruebas correspondientes a las HU 1 y 2, los que arrojaron 18 no conformidades las cuales fueron subsanadas en su totalidad, en la segunda iteración se aplicaron 22 pruebas correspondientes a las HU 3 y 4, en los mismos se encontraron 12 no conformidades que fueron resueltas, en la tercera iteración se aplicaron 18 pruebas correspondientes a las HU 5 y 6 encontrándose 5 no conformidades las cuales se corrigieron y en la cuarta y última iteración se aplicaron 14 pruebas correspondientes a las HU 7, 8, 9 y 10, en las que no se encontraron no conformidades. El desarrollo de las pruebas de aceptación contribuyó a validar que la aplicación tenga la calidad requerida y esté libre de errores que podrían causar insatisfacciones en el cliente.

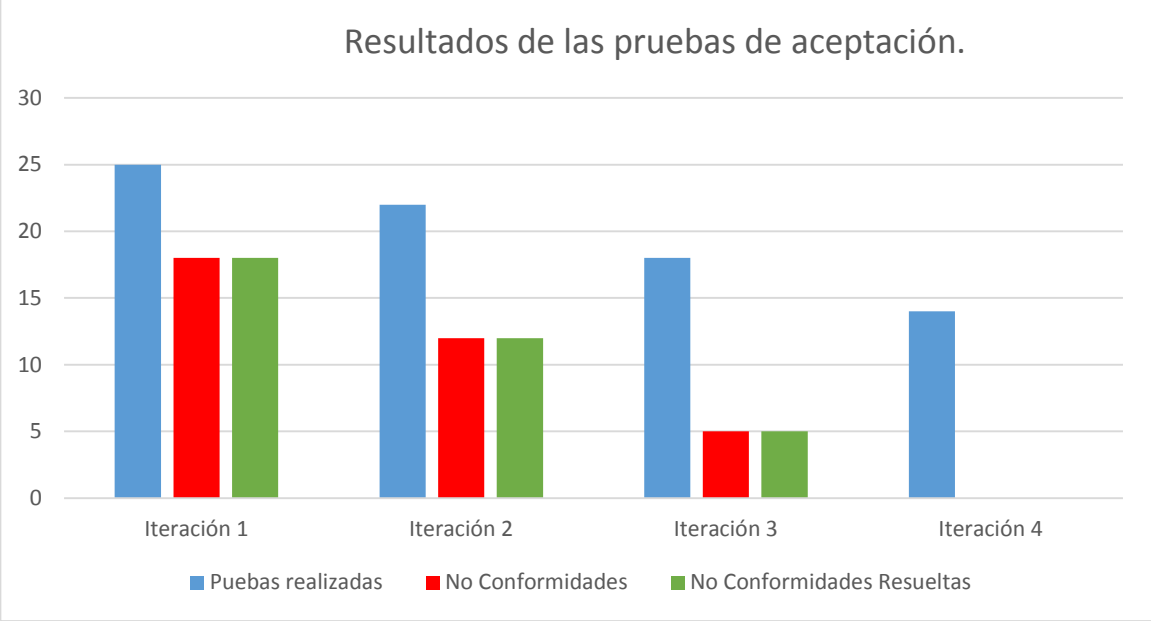


Ilustración 18 Resultados de las pruebas de aceptación.
Fuente: Elaboración propia.

3.5.2 Pruebas Unitarias.

Una prueba unitaria es la verificación de un módulo (unidad de código) determinado dentro de un sistema. El concepto de "módulo" varía de acuerdo al lenguaje de programación que se esté utilizando; por ejemplo, en Java sería una clase. Las pruebas unitarias aseguran que un determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema. (31)

En XP los programadores deben escribir las pruebas unitarias para cada módulo antes de escribir el código. No es necesario escribir casos de prueba para todos los módulos, sólo para aquellos en que exista la posibilidad de que puedan fallar. Luego de escribir el código, los programadores ejecutan las pruebas, las cuales deben resultar 100% efectivas para que el código pueda integrarse al sistema. En caso contrario hay que solucionar los errores y ejecutar nuevamente los casos de prueba hasta verificar que no exista ninguno de ellos. Las pruebas son automatizadas utilizando herramientas como xUnit, de forma tal de poder soportar un testing continuo y mantener organizados los casos de pruebas. (31)

Para el desarrollo de estas pruebas se utilizó el framework JUnit, ya que se trata de un marco de trabajo de código abierto para la automatización de las pruebas de unidad de aplicaciones Java en los proyectos de

software. Provee al usuario de herramientas, clases y métodos que facilitan la tarea de realizar pruebas en el sistema y así asegurar su consistencia y funcionalidad. (32)

El proceso de pruebas de caja blanca se concentró principalmente en validar que cada segmento de código funcione apropiadamente. Además, cuenta con una interfaz simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada. Este tipo de prueba es totalmente objetiva y por lo tanto puede realizarla un ordenador de forma repetitiva, no depende de la experiencia del programador (32). Para hacer uso de este marco de trabajo se definieron los siguientes pasos:

Paso 1. Definición de los métodos a probar.

Las pruebas se realizaron solo a las funcionalidades que determinan el correcto funcionamiento del sistema, entre ellas están:

Tabla 12: Métodos para las pruebas unitarias

Funcionalidades	
+adicionarActivo () **+ eliminarActivo ()** **+ modificarActivo ()** **+ validarSiExiste ()** **+ listarActivos ()** **+ adicionarRolUsuario ()** **+ eliminarRolUsuario ()** **+ modificarRol ()** **+ listarUsuarios ()** **+ adicionarComponente ()** **+ eliminarComponente ()** **+ modificarComponente ()** **+ listarComponentes ()** **+ adicionarDependenciaExterna ()** **+ eliminarDependencia ()**	**+ modificarDependencia ()** **+ listarDependencias ()** **+ adicionarAmenaza ()** **+ eliminarAmenaza ()** **+ modificarAmenaza ()** **+ listarAmenaza ()** **+ adicionarDiagrama ()** **+ eliminarDiagrama ()** **+ modificarDiagrama ()** **+ listarDiagramas ()** **+ adicionarComponenteBotton ()** **+ buscarDiagrama ()** **+ crearProyectoNuevo()** **+ accederProyecto()** **+ eliminarProyectoCompleto()**

Paso 2. Adaptación de los métodos al framework JUnit

Para explicar el proceso de realización de las pruebas de Caja Blanca con JUnit, fue preciso realizar algunas modificaciones que no cambian la lógica del negocio. Asimismo es necesario definir que los métodos testados sean de tipo void, identificar juegos de datos para probar cada método y agregar en la parte superior del método la anotación @Test. (32) Se tomó como objeto de estudio el método adicionarActivo (), función que se encuentra en la clase GestionarActivoImp y que permite adicionar correctamente una activo en el sistema, (ver Ilustración 19).

```
@Test
public void adicionarActivo() throws Exception {

    String nombreaActivo = "Activo1";
    String descripcion = "descripcion";
    String paranoia = "paranoia";

    if (!activoDAO.existeActivoEnProyecto(nombreaActivo, 1)) {
        Activo a = new Activo(nombreaActivo, paranoia, descripcion,(Proyecto) observado.getProyecto());
        if (idActivo != null) {
            a.setId(1);
        }
        save(a);

    } else {
        throw new Exception("El activo ya existe.");
    }
}
```

Ilustración 19 : Código generado con JUnit en Netbeans
Fuente: Elaboración Propia.

Paso 3. Realización de las pruebas

Los resultados de las pruebas son almacenados dentro de una lista. Luego, el marco de trabajo JUnit comprueba que cada uno de los resultados obtenidos coincide con los resultados esperados y muestra en una ventana los resultados obtenidos, cuando estos son satisfactorios para todas las pruebas realizadas, se observa una línea verde en la ventana, en caso contrario aparece una línea roja (32).

Se desarrollaron un total de 27 pruebas con el JUnit, distribuidas en dos iteraciones. Se obtuvieron en la primera un total de 5 errores que fueron subsanados. En la última se obtuvieron resultados satisfactorios para cada prueba realizada. En la Ilustración 20 aparecen los resultados de las pruebas.

Ilustración 20: Resultados de las pruebas con JUnit
Fuente: Elaboración propia.

3.6 Conclusiones parciales.

Luego de concluido el presente capitulo se puede afirmar haber cumplido con las tareas planificadas para este. Fueron especificados para apoyar el proceso de implementación, las tarjetas CRC para la representación de las clases y la colaboración entre ellas logrando hacer lo más claro posible el flujo de los datos en el sistema. Se enuncian los patrones de diseño que validan que exista la menor dependencia posible entre las clases de sistema pero a la vez toda trabajen sobre la misma área de trabajo y que cada clase lleve a cabo las funciones para las que fue creada. Se llevó a cabo la implementación de la solución propuesta utilizándose para esto los estándares de codificación propuestos para asegurar que el código sea legible para futuras actualizaciones de la herramienta. Se aplicaron finalmente las pruebas de aceptación y las pruebas unitarias para validar los resultados obtenidos en la fase de implementación. Estas pruebas permitieron corregir una serie de funcionalidades las cuales fueron subsanadas, permitiendo que el sistema posea un correcto funcionamiento.

CONCLUSIONES GENERALES

Al ser seleccionada la metodología de desarrollo XP para la implementación de la solución propuesta se describen y especifican sus respectivos artefactos con el fin de plasmar las principales características de la nueva versión a desarrollar, entre las que se encuentran las historias de usuario para especificar los requerimientos del cliente, las tareas de ingeniería para describirlos con mayor nivel de detalle y las tarjetas CRC para mostrar las relaciones entre las principales clases de la aplicación. La utilización del patrón n capas en conjunto con varios patrones de diseño contribuyó a una mejor organización del código, facilitando el trabajo a la vez que resolvió varias interrogantes sobre las acciones a acometer para dar solución a problemas como la dependencia entre las clases y la asignación de las responsabilidades a estas. Finalmente gracias a la aplicación de un conjunto de pruebas es comprobado el buen funcionamiento del software.

Luego de finalizada la presente investigación se puede concluir que con la implementación de la versión 2.0 de la herramienta Segursoft quedan resueltas las limitaciones que presentaba para la aplicación de nuevas técnicas para el análisis y gestión de riesgos de seguridad en el Centro TLM. Para este fin se hizo una investigación de varias herramientas que implementan dichas técnicas, de las cuales, luego de confirmar que no es posible su utilización para resolver la problemática planteada, se tomaron aportes para incluir en la solución propuesta. Estos aportes son la técnica propuesta por MAGERIT para evaluar el nivel de riesgo de una amenaza, las técnicas para el modelado de amenazas (representadas mediante los diagramas de grafos de ataque, casos de mal uso y diagrama superficial de activos) y las técnicas para la descripción de casos de mal uso y especificación de requisitos de seguridad a través de plantillas creadas para estandarizar dichos procesos.

RECOMENDACIONES

- Incluir técnicas para el análisis cuantitativo del riesgo que supone una determinada amenaza.
- Permitir la importación de diagramas creados en la herramienta Visual Paradigm.

REFERENCIAS BIBLIOGRÁFICAS

1. **Viruslist.** Viruslist.com. *Todo sobre seguridad en internet.* [En línea] 5 de 12 de 2013. [Citado el: 1 de 4 de 2015.] http://www.viruslist.com/sp/analysis?pubid=207271238.
2. **Yeja, Ing. Adrian Hernández.** *Estrategia para la detección de vulnerabilidades en aplicaciones web durante la fase de implementación del ciclo de desarrollo de software.* La habana : UCI, 2014.
3. **Mecias, Lilian Teresa Castro.** *Guía de gestión del riesgo tecnológico para el tratamiento de la seguridad durante el proceso de desarrollo de software.* La Habana : UCI, 2014.
4. **McGraw, Gary.** *Software Security: Building Security In.* Boston : Addison Wesley Professional, 2006.
5. **Marta Castellaro, Susana Romaniz, Juan Carlos Ramos,Carlos Feck, Ivana Gaspoz.** *Aplicar el Modelo de Amenazas para incluir la Seguridad.* Santa Fe : Universidad Tecnológica Nacional, 2008.
6. **Ivar Jacobson, Grady booch, James Rumbaugh.** *El proceso unificado de desarrollo de software.* Madrid : Pearson Educación, S.A, 2000.
7. **Zepeda, Vianca Vega.** *Ingeniería de Requerimientos para Productos Seguros.* Antofagasta : Unversidad Católica del Norte, 2006.
8. **Gómez, Miguel Ángel Amutio.** *Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información.* Madrid : Ministerio de Hacienda y Administraciones Públicas, 2012.
9. **José A. Mañas, Carlos Belso.** *Gestión Dinámica de Riesgos: Seguridad de la Red de Servicios.* Madrid : Universidad Politécnica de Madrid, 2013.
10. **Security, Department of Homeland.** Build Security In. *Setting a higher standard for software assurance.* [En línea] Homeland Security, 18 de 9 de 2008. [Citado el: 20 de 4 de 2015.] https://buildsecurityin.us-cert.gov/bsi/.
11. **Mellado, Daniel.** *Un Proceso de Ingeniería de Requisitos de Seguridad en la Práctica.* Madrid : IEEE Latin America, 2007.
12. **Firesmith, Donald.** *Security Use Cases.* U.S.A : Journal of Object Technology, 2003.
13. **Microsoft.** Aplicación de Microsoft amenaza Modeling Blog. *Análisis y modelado (TAM) v3.0.* [En línea] Microsoft Corporación, 20 de 7 de 2009. [Citado el: 18 de 5 de 2015.] http://blogs.msdn.com/b/threatmodeling/archive/2009/07/20/threat-analysis-and-modeling-tam-v3-0-learn-about-the-new-features.aspx.
14. **P.F, Daniel.** *Análisis y Modelado de Amenazas.* Asturias : -, 2006.
15. **Laebel, Gabi.** Practical Threat Analysis for Information Security Experts. *PTA Technologies.* [En línea] Eldan Software Systems Ltd., - de - de 2013. [Citado el: 15 de 5 de 2015.] http://www.ptatechnologies.com/default.htm.
16. **Flores, Lic. Ervin.** Ingenieria de Software. [En línea] Universidad Bolivariana, - de - de 2015. [Citado el: 3 de 4 de 2015.] http://ingenieriadesoftware.mex.tl/52753_XP---Extreme-Programing.html.
17. **Reyna, Rafael.** Scribd. *Librería del mundo digital.* [En línea] Attribution Non-commercial, 23 de 5 de 2013. [Citado el: 6 de 4 de 2015.] https://es.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.
18. **Paradigm, Vsual.** Manual Online. [En línea] Visual Paradim, 23 de 12 de 2014. [Citado el: 8 de 4 de 2015.] http://www.visual-paradigm.com/features/.
19. **NetBeans.** Gembeta Dev. [En línea] 9 de 1 de 2014. [Citado el: 15 de 4 de 2015.] http://www.genbetadev.com/herramientas/netbeans-1.

20. **Cuba.** Ecured. *Enciclopedia Cubana.* [En línea] 14 de 2 de 2010. [Citado el: 10 de 4 de 2015.] http://www.ecured.cu/index.php/Lenguaje_de_programaci%C3%B3n_Java.
21. **González, Héctor Suárez.** *Manual Hibernate.* - : Documentación de javaHispano, 2013.
22. **Andalucia, Junta de.** Marco de Desarrollo. *Hibernate.* [En línea] ©Junta de Andalucia, 1 de 3 de 2013. [Citado el: 19 de 5 de 2015.] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/97.
23. **Lorena Sanchez .** Que es Microsoft Visio y sus Características. [En línea] 13 de 1 de 2013. [Citado el: 2 de 6 de 2015.] http://lorehidal.blogspot.com/.
24. **IDEF0 Integrated Definition Methods.** *IDEF0.* [En línea] KBSI Community, 2010. [Citado el: 1 de 6 de 2015.] http://www.idef.com/IDEF0.htm.
25. **Martinez, Rafael.** PostgreSQL-es. *Sobre PostgreSQL.* [En línea] Creative Commons, 2 de 10 de 2010. [Citado el: 20 de 5 de 2015.] http://www.postgresql.org.es/sobre_postgresql.
26. **Álvarez, Cecilio.** Genbeta:dev. *Desarrollo y Software.* [En línea] -, 12 de 8 de 2014. [Citado el: 12 de 4 de 2015.] http://www.genbetadev.com/frameworks/que-es-spring-framework.
27. **Leibys Oria Pardo, José Miguel Noa Cobas.** *Herramienta para el Análisis y Modelado de Amenazas en sistemas informáticos.* La Habana : UCI, 2012.
28. **Gestión de proyectos y desarrollo de software.** *Desarrollo de software. Tarjetas CRC.* [En línea] Jummp, 10 de 1 de 2012. [Citado el: 12 de 5 de 2015.] https://jummp.wordpress.com/2012/01/10/desarrollo-de-software-tarjetas-crc/.
29. **Microsoft. Developer Network.** *Revisiones de código y estándares de codificación.* [En línea] Microsoft, - de - de 2003. [Citado el: 2 de 5 de 2015.] https://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.
30. **López, Denys López.** *Desarrollo de una herramienta para el diseño gráfico de políticas de seguridad.* La Habana : UCI, 2012.
31. **Cukerman, Diego.** *Testing en eXtreme Programming.* Uruguay : Universidad de la República - Uruguay, 2006.
32. **Junit.** [En línea] 4 de 12 de 2014. [Citado el: 25 de 5 de 2015.] http://junit.org/.

ANEXOS

ANEXO I Historias de usuario.

Historia de usuario Nro.1: Cambiar método de almacenamiento de los datos, de ficheros a base de datos.

Tabla 13 Historia de usuario #8

Historia de usuario	
Número: **1**	**Usuario:** Analista.
Nombre de Historia de Usuario: Cambiar método de almacenamiento de los datos, de ficheros a base de datos.	
Prioridad en negocio: Alto.	**Riesgo en Desarrollo:** Alto.
Puntos estimados: 2	**Iteración Asignada:** 1
Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema permitirá al usuario guardar en una base de datos la información que se gestione dándole mayor seguridad a esta.	
Observaciones:	
Prototipo de interfaz:	

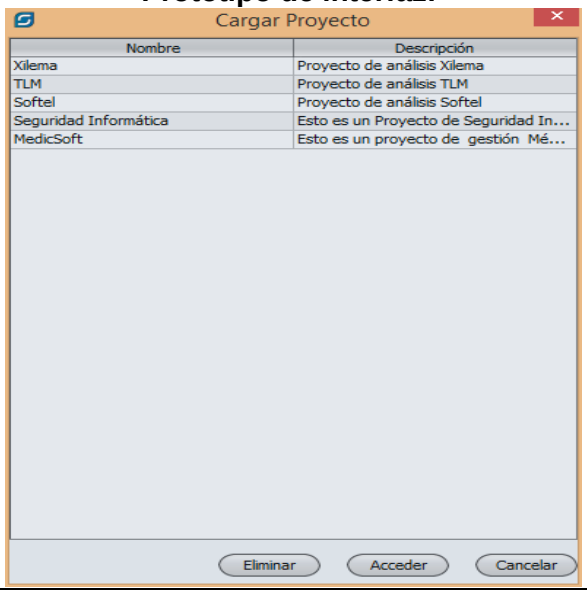
Historia de usuario Nro.2: Acceder a un proyecto de análisis.

Tabla 14 Historia de usuario #2

Historia de usuario	
Número: 2	**Usuario:** Analista.
Nombre de Historia de Usuario: Acceder a un proyecto de análisis.	
Prioridad en negocio: Alta.	**Riesgo en Desarrollo:** Alto.

Puntos estimados: 1	**Iteración Asignada:** 1
Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema permitirá al usuario cargar un proyecto de análisis exístete en la base de datos brindando la posibilidad de seleccionarlo de una lista que debe mostrar con todos los proyectos existentes. También debe brindar la posibilidad de eliminar un proyecto deseado la lista.	
Observaciones:	

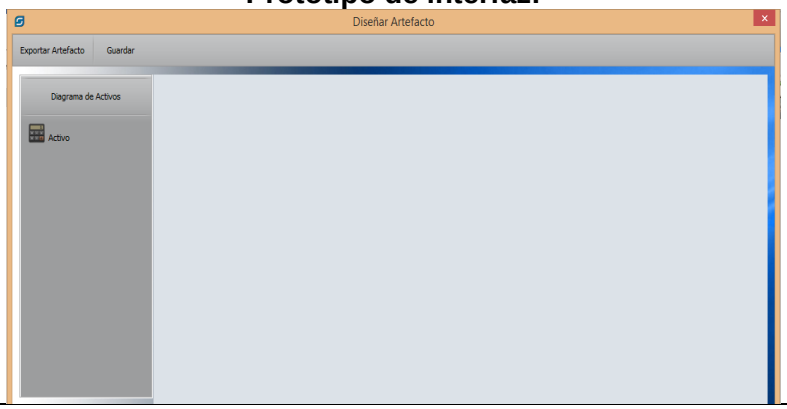
Prototipo de interfaz:



Historia de usuario Nro.4: Diseñar diagrama superficial de activos.

Tabla 15 Historia de usuario #4

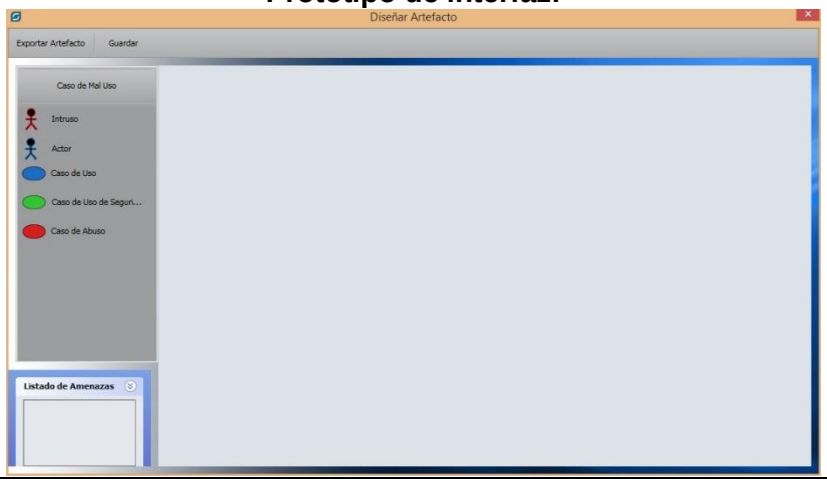
Historia de usuario	
Número: 4	**Usuario:** Analista.
Nombre de Historia de Usuario: Diseñar diagrama superficial de activos.	
Prioridad en negocio: Medio.	**Riesgo en Desarrollo:** Medio.

Puntos estimados: 2	**Iteración Asignada:** 2
Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema debe permitir el modelado del artefacto ¨Diagrama Superficial de Activos¨. Para ello debe mostrar una interfaz con los estereotipos correspondientes a este diagrama y un área para crear el diagrama. Debe permitir exportar el diagrama modelado como imagen.	
Observaciones:	
Prototipo de interfaz: 	

Historia de usuario Nro.5: Diseñar diagrama de mal uso.

Tabla 16 Historia de usuario #5

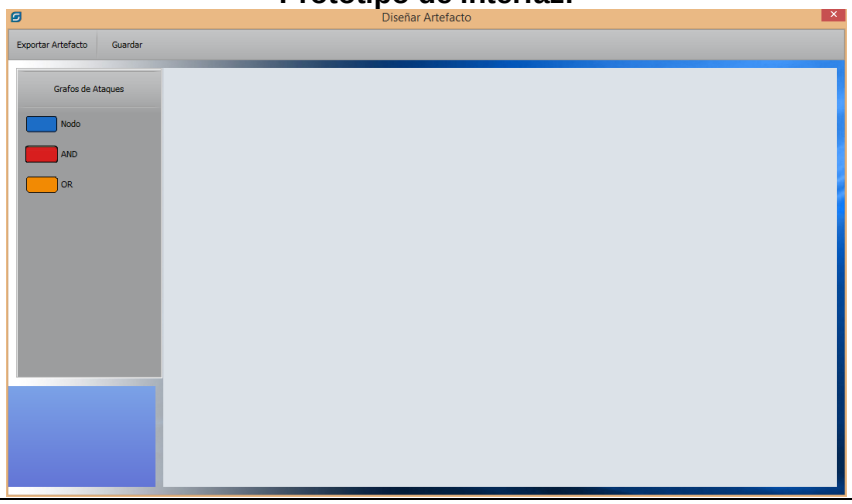
Historia de usuario	
Número: 5	**Usuario:** Analista.
Nombre de Historia de Usuario: Diseñar diagrama de mal uso.	
Prioridad en negocio: Media.	**Riesgo en Desarrollo:** Medio.
Puntos estimados: 2	**Iteración Asignada:** 3

Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.
Descripción: El sistema debe permitir el modelado del artefacto ¨Diagrama de Mal Uso¨. Para ello debe mostrar una interfaz con los estereotipos correspondientes a este diagrama y un área para crear el diagrama. Debe permitir generar un reporte que contenga los casos de mal uso y los requisitos de seguridad especificados en el diagrama y permitir exportarlo en formato .pdf. Finalmente debe brindar la opción de exportar el diagrama modelado como imagen.
Observaciones:
Prototipo de interfaz: 

Historia de usuario Nro.6: Diseñar grafo de ataque.

Tabla 17 Historia de usuario #6

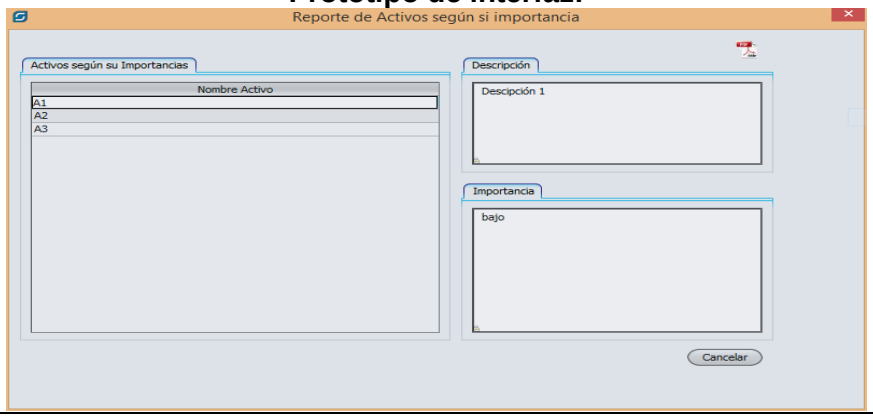
Historia de usuario	
Número: 6	**Usuario:** Analista.
Nombre de Historia de Usuario: Diseñar grafo de ataque.	
Prioridad en negocio: Media.	**Riesgo en Desarrollo:** Medio.
Puntos estimados: 2	**Iteración Asignada:** 3

Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.
Descripción: El sistema debe permitir el modelado del artefacto ¨Grafo de Ataque¨. Para ello debe mostrar una interfaz con los estereotipos correspondientes a este diagrama y un área para crear el diagrama. Debe permitir exportar el diagrama modelado como imagen.
Observaciones:
Prototipo de interfaz: 

Historia de usuario Nro.7: Generar reporte de activos según su Importancia.

Tabla 18 Historia de usuario #7

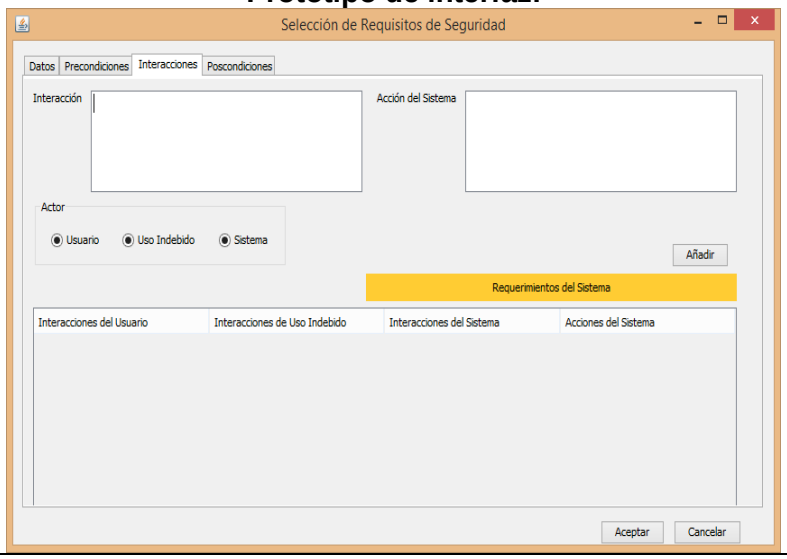
Historia de usuario	
Número: 7	**Usuario:** Analista.
Nombre de Historia de Usuario: Generar reporte de activos según su Importancia.	
Prioridad en negocio: Media.	**Riesgo en Desarrollo:** Alto.

Puntos estimados: 1	**Iteración Asignada:** 4
Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema permite al usuario generar un reporte de activos según su importancia, la aplicación muestra una ventana donde el usuario podrá observar una lista de todos los activos con sus características con la posibilidad de exportar dicho reporte como PDF.	
Observaciones:	
Prototipo de interfaz: 	

Historia de usuario Nro.8: Especificar requisitos de casos de uso de seguridad.

Tabla 19 Historia de usuario #8

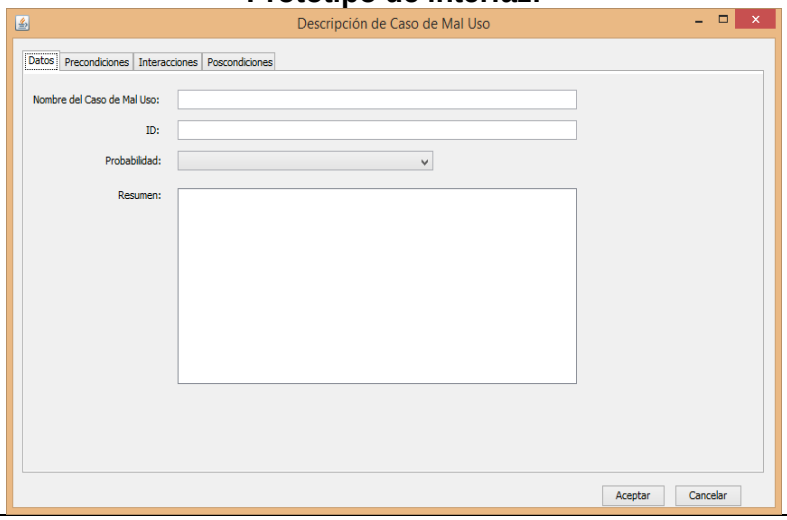
Historia de usuario	
Número: 8	**Usuario:** Analista.
Nombre de Historia de Usuario: Especificar requisitos de casos de uso de seguridad.	
Prioridad en negocio: Media.	**Riesgo en Desarrollo:** Media.
Puntos estimados: 1	**Iteración Asignada:** 4

Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema permite al usuario especificar los requisitos de seguridad pertenecientes a los casos de uso de seguridad, para ello se deben llenar los campos de una plantilla establecida.	
Observaciones:	
Prototipo de interfaz: 	

Historia de usuario Nro.9: Especificar descripción de casos de mal uso.

Tabla 20 Historia de usuario #9

Historia de usuario	
Número: 9	**Usuario:** Analista.
Nombre de Historia de Usuario: Especificar descripción de casos de mal uso.	
Prioridad en negocio: Media.	**Riesgo en Desarrollo:** Media.
Puntos estimados: 1	**Iteración Asignada:** 4

Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.	
Descripción: El sistema permite al usuario especificar la descripción de un nuevo caso de mal uso, para ello se deben llenar los campos de una plantilla establecida.	
Observaciones:	
Prototipo de interfaz: 	

Historia de usuario Nro.10: Generar Reporte de requisitos de seguridad en formato pdf.

Tabla 21 Historia de usuario #10

Historia de usuario	
Número: 10	**Usuario:** Analista.
Nombre de Historia de Usuario: Generar Reporte de requisitos de seguridad en formato pdf.	
Prioridad en negocio: Baja.	**Riesgo en Desarrollo:** Media.
Puntos estimados: 1	**Iteración Asignada:** 4

Programador responsable: Addiel Velazco Padilla – Yordanis Calvo Rodríguez.
Descripción: El sistema permite al usuario obtener un reporte en formato .pdf conteniendo una lista con todos los casos de mal uso y sus respectivos requisitos de seguridad, los cuales fueron especificados anteriormente por dicho usuario.
Observaciones:
Prototipo de interfaz: