Facultad 4

Sistema para el Análisis y Gestión de Riesgos de Seguridad Informática en la Facultad 4

Autor:

Roides Javier Cruz Lara

Tutor:

Ing. Renier Portelles Cobas

Junio 2015

“La felicidad existe sobre la tierra; y se conquista con el ejercicio prudente de la razón, el conocimiento de la armonía del universo, y la práctica constante de la generosidad. Ser bueno es el único modo de ser dichoso. Ser culto es el único modo de ser libre. Pero, en lo común de la naturaleza humana, se necesita ser próspero para ser bueno.”

AGRADECIMIENTOS

Mi primer agradecimiento va hacia mis padres Lurdes y Roberto por hacerme el hombre que soy, por inculcarme el interés por el conocimiento y la ética, pero sobre todo por creer en mí.

A mis dos hermanos por su cariño y apoyo incondicional, por darme tantas alegrías.

A mis amigos viejos y nuevos que siempre están a mi lado sin importar la distancia y son la familia que me ha dado la vida.

A mi tutor Portelles por su sabiduría, paciencia, comprensión e inteligencia.

A Wendy por esperarme.

A mis padres……

A mis hermanos……

A mis amigos……

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del presente trabajo titulado: “Sistema para el Análisis y Gestión de Riesgo de Seguridad Informática en la Facultad 4” y autorizamos a la Universidad de las Ciencias Informáticas y al Departamento de Tecnología de la Facultad a usarlo en su beneficio.

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

Roides Javier Cruz Lara	**Ing. Renier Portelles Cobas**
Autor	**Tutor**

RESUMEN

La Gestión de Riesgos permite determinar cómo es, cuánto vale y cómo de protegido se encuentra un sistema de información para luego, elaborar un plan de seguridad que, implantado y operado, satisfaga los objetivos propuestos con el nivel de riesgo que acepta la dirección de una organización . Esta investigación muestra los resultados obtenidos de la implementación de un Sistema para el Análisis y Gestión de Riesgos de Seguridad Informática en la Facultad 4, cuyo objetivo es reducir la complejidad del proceso de Gestión de Riesgos automatizando tareas como la identificación de activos no esenciales, facilitando la valoración y clasificación de los activos e identificando amenazas de seguridad reportadas en las incidencias de GLPI.

PALABRAS CLAVE

Activo, amenaza, salvaguarda, riesgo.

ABSTRACT

Risk Management is the process of Information Security that determine how, and how much is protected an information system, so the security plan can be developed, implemented and operated, in order to reach the proposed objectives at the acceptance risks level of the organization. This research shows the results of the System for Analysis and Risk Management Information Security implementation at the Faculty 4. This software aims to reduce the complexity of Risk Management by automating tasks such as the identification of non-core assets, facilitating the valuation and classification of assets and identifying security threats reported in GLPI incidents.

KEYWORDS

Asset, threat, safeguard, risk.

ÍNDICE

INTRODUCCIÓN 1
CAPITULO 1. FUNDAMENTACIÓN TEÓRICA 6
1.1 Introducción 6
1.2 Conceptos asociados 6
1.2.1 Riesgo de seguridad informática 6
1.2.2 Análisis de riesgos 6
1.2.3 Gestión de riesgos 6
1.2.4 Gestión de Riesgo en el Sistema de Gestión de Seguridad de la Información 7
1.3 Principales estándares de seguridad y metodologías para la Gestión de Riesgos 7
1.3.1 ISO/IEC 27000 7
1.3.2 ISO/IEC 27001 8
1.3.3 ISO/IEC 27005 9
1.3.5 NIST SP 800-30 9
1.3.6 OCTAVE 10
1.3.4 Selección de la metodología MAGERIT 10
1.4 Análisis de soluciones similares 11
1.4.1 R-Box 11
1.4.2 PILAR 13
1.4.3 Resultados del análisis de soluciones 14
1.5 Metodologías de desarrollo de software 14
1.5.1 Extreme Programing (XP) 14
1.5.2 Scrum 15
1.5.3 Rational Unified Process (RUP) 15
1.5.4 Resultados del análisis de las metodologías de desarrollo de software 16
1.6 Herramientas y lenguajes de programación 17
1.6.1 Herramienta de modelado Visual Paradigm 17
1.6.2 Lenguaje de Modelado Unificado 17
1.7 Tecnología del lado del servidor. 17
1.7.1 Lenguaje PHP 17
1.7.2 Servidor web Apache 18
1.8 Tecnologías del lado del cliente 18

1.8.1 Hyper Text Markup Language. HTML5 18
1.8.2 Javascript 19
1.8.3 Cascading Style Sheets (CSS) 19
1.9 Entorno de desarrollo integrado (IDE) NetBeans. 19
1.10 Framework de desarrollo 20
1.10.1 Symfony 2.3.5 20
1.10.2 Bootstrap 3 20
1.11 Sistema gestor de bases de datos MySQL 21
1.12 Propuesta y selección de las herramientas 21
1.13 Conclusiones del capítulo 22
CAPÍTULO 2 PROPUESTA DE SOLUCIÓN 23
2.1 Introducción 23
2.2 Introducción a MAGERIT-versión 3.0 23
2.3 Propuesta de solución 24
2.3.1 El valor de los activos 25
2.3.2 Jerarquía y dependencias entre activos 25
2.3.3 El valor e impacto acumulado 26
2.3.4 Impacto repercutido 26
2.3.5 Paquete de salvaguardas 27
2.3.6 Riesgo 27
2.4 Usuarios del sistema 27
2.5 Lista de funcionalidades del sistema 28
2.6 Restricciones del sistema 30
2.7 Fase de Exploración 30
2.7.1 Historias de Usuarios. 31
2.8 Fase de Planificación 32
2.8.1 Estimación del esfuerzo por HU 32
2.8.2 Plan de Iteraciones 34
2.8.3 Plan de entrega 36
2.9 Tarjetas de Colaboración y Responsabilidad de Clases 37
2.10 Prototipo de interfaz de usuario 38
2.11 Conclusiones del capítulo. 40
CAPITULO 3. IMPLEMENTACIÓN Y PRUEBAS 41

3.1 Introducción 41
3.2 Arquitectura de Software 41
3.2.1 Patrón Modelo Vista Controlador 41
3.3 Patrones de diseño GRASP 42
3.4 Modelo de Datos. 43
3.5 Tareas de ingeniería 44
3.6 Pruebas 45
3.6.1 Pruebas unitarias 46
3.6.2 Pruebas de aceptación 47
3.6.3 Resultados de las pruebas 49
3.7 Evaluación de beneficios 50
3.8 Conclusiones del capítulo 53
CONCLUSIONES 55
RECOMENDACIONES 56
REFERENCIAS 57
ANEXOS 60

INTRODUCCIÓN

La información que posee y procesa una organización es un valioso activo que permite alcanzar objetivos de negocio. En la actualidad la mayor parte de la información reside en equipos informáticos, soportes de almacenamiento, redes de datos y en general tecnologías de la información y las comunicaciones (TIC) que engloban los sistemas de información de las organizaciones. Estos sistemas están sujetos a riesgos inherentes a su uso, pueden presentar vulnerabilidades, las cuales, de no ser controladas, podrían materializar amenazas que generen un impacto negativo en la organización. Por esta razón la necesidad del aseguramiento del valor de TIC, la administración de los riesgos asociados, así como el incremento de requerimientos para controlar la información, devienen elementos clave para la administración de una empresa u organización de cualquier índole (1).

La Seguridad de la Información[1] es el conjunto de métodos y herramientas destinados a la preservación de la confidencialidad, integridad y disponibilidad de información de las instituciones, ésta abarca un importante número de disciplinas y especialidades distintas y complementarias que deben ser gestionadas (2). Existen diferentes modelos, estándares, recomendaciones y regulaciones relacionados con la gestión de la Seguridad de la Información. Entre ellos se destaca la serie de estándares ISO/IEC 27000, que poseen gran aceptación a nivel internacional y es posible certificar. La cantidad y diversidad de controles que es necesario implementar, hace que la gestión de la Seguridad Informática sea un proceso complejo (3).

En la Universidad de las Ciencias Informáticas (UCI) se han realizado diferentes trabajos en pos de lograr una mayor eficiencia e integración entre los diferentes procesos de gestión de la Seguridad Informática. En este contexto se destaca el Modelo para la Gestión Automatizada e Integrada de Controles de Seguridad Informática (GAISI) propuesto por Raydel Montesinos Perurena. El modelo GAISI define diez macro-controles que representan una agrupación y síntesis de los controles automatizables identificados en los estándares ISO/IEC 27002 y NIST SP 800-53, donde la automatización es aplicada a las acciones de operación, monitorización y revisión pertenecientes a las fases Hacer y Verificar del Sistema Gestión de Seguridad de la Información (SGSI) definido por ISO/IEC 27001 (3). En la Facultad 4 la Dirección de Tecnología ha implementado diferentes sistemas que permiten una gestión más eficiente de la Seguridad

[1] La presente investigación considerará los términos “seguridad de la información”, “seguridad informática” equivalentes, acogiéndose a la Resolución 127/2007 del Ministerio de la Informática y las Comunicaciones de Cuba (43).

Informática como el Control del Inventario de Activos y la Gestión de Incidentes para el soporte técnico. Sin embargo, todavía se presentan deficiencias en la fase Planificar, específicamente en el proceso de Gestión de Riesgos de Seguridad Informática. Actualmente este proceso se realiza mediante una hoja de cálculo Microsoft Excel que permite introducir algunos datos y calcular los riesgos asociados a los activos individualmente. Esta forma de proceder conlleva a que el proceso de Análisis de Riesgos sea engorroso y complejo debido al número de activos que presenta la Facultad 4, y la variabilidad de sus estados. Frecuentemente se cometen errores en la información introducida y posterior error en el análisis de riesgos. El tiempo que se necesita para introducir toda la información de los activos es extremadamente largo. Se suma a lo antes planteado, la dificultad de contar con información actualizada sobre los activos para lograr un análisis de riesgos veraz y preciso. El estado actual de este proceso no admite el nivel integración con el resto de los procesos de gestión de Seguridad Informática ya automatizados por parte de la Dirección de Tecnología de la Facultad 4. Para la realización de esta tarea existen algunas herramientas pero son propietarias, por tanto no se corresponden con la política de la UCI acerca del uso de software libre. Teniendo en cuenta lo antes planteado se identificó el siguiente **problema científico**: ¿Cómo disminuir la complejidad del proceso de Gestión de Riesgos de Seguridad Informática en la Facultad 4?

Desde el punto de vista cuantitativo la complejidad de un sistema está dada por la cantidad de información necesaria para describirlo, así como el nivel y multitud de dependencias entre los diferentes elementos que lo conforman y el nivel de previsibilidad de sus estados (4). Los sistemas de información empresariales tienden a ser complejos, esto hace que los procesos de gestión que se realizan sean, por tanto, procesos complejos. Extrapolando estos conceptos, la complejidad de un proceso puede ser definida entonces (cuantitativamente) como la cantidad de variables o elementos dependientes entre sí que procesa, la cantidad de información necesaria para realizarlo y en general puede ser proporcional al nivel de complejidad del sistema que gestiona. Esta complejidad se produce por la existencia e interacción de:

1. Las variables involucradas y sus interacciones y por consiguiente las dimensiones de la situación.
2. Los diferentes estados que se pueden producir.
3. La dinámica de las relaciones entre las variables.
4. El tiempo de que se dispone para atender una situación determinada.
5. La centralización y dependencia para atender la situación.

6. La información disponible.
7. La velocidad de adaptación requerida.
8. La desviación que existe entre ejecutar actividades orientadas a la misión y desviadas de ella (5).

Una variante efectiva para lograr que la Gestión de Riesgos de Seguridad Informática sea un proceso menos complejo, en un entorno tecnológico amplio y en constante cambio es elevar el nivel de informatización del proceso. Para solucionar el problema planteado se considera como **objeto de estudio** el proceso de Gestión de la Seguridad Informática en la Facultad 4, y como **campo de acción** la Gestión de Riesgos de Seguridad Informática en la Facultad 4.

El **objetivo general** de la presente investigación es desarrollar un sistema informático para el Análisis y Gestión de Riesgos de Seguridad Informática, que contribuya a disminuir la complejidad de este proceso en la Facultad 4.

Para alcanzar el objetivo general se plantean los siguientes **objetivos específicos**:

1. Analizar los referentes teóricos en que se fundamenta la Gestión de la Seguridad Informática y el proceso Gestión de Riesgos, las técnicas y herramientas existentes.
2. Determinar las tareas de Gestión de Riesgos a informatizar.
3. Implementar el Sistema para el Análisis y Gestión de Riesgo de Seguridad Informática.
4. Realizar las pruebas que garanticen la calidad del software desarrollado.

Para cumplir los objetivos trazados se plantean las siguientes **tareas de la investigación**:

1. Investigar la base teórica y conceptos de Gestión de Seguridad Informática y Gestión de Riesgos de Seguridad Informática.
2. Realizar un análisis de las principales metodologías, técnicas y herramientas para la Gestión de Riesgo en seguridad informática.
3. Seleccionar las herramientas y tecnologías a utilizar en el desarrollo del el Sistema para el Análisis y Gestión de Riesgo.
4. Definir qué tareas de gestión de riesgos se informatizarán.
5. Realizar levantamiento de los requisitos del Sistema para el Análisis y Gestión de Riesgo.

6. Realizar análisis y diseño del sistema.
7. Implementar el Sistema para el Análisis y Gestión de Riesgo.
8. Validar sistema mediante la aplicación de pruebas de software.

Como **hipótesis de investigación** se plantea que: el desarrollo de un sistema para el Análisis y Gestión de Riesgo de Seguridad Informática, basado en sistemas de gestión de información, disminuirá la complejidad del proceso de Gestión de Riesgos de Seguridad Informática en la Facultad 4 mediante la disminución del tiempo que se dispone para realizar este proceso y el aumento de la velocidad de adaptación ante los cambios en el sistema de información.

Métodos Teóricos: El método **analítico-sintético** se aplica para analizar la documentación referente al Análisis de Riesgos, las diferentes tecnologías y herramientas que se utilizan en este entorno, para luego definir una síntesis que sirva de marco teórico a la investigación y lograr una mejor comprensión del objeto de estudio. El método **histórico-lógico** y el **dialéctico** son utilizados para realizar un estudio crítico de los trabajos previamente realizados que abordan esta materia.

Métodos Empíricos: Se utiliza **análisis documental** en la revisión de la literatura especializada, el **experimental** y la **medición** para la validación de los resultados de la investigación. Se emplea además la **entrevista** para la consulta de expertos y la valoración del posible impacto de la solución propuesta en el Sistema la Gestión de la Seguridad Informática de la Facultad 4.

La estructura del trabajo comprende 3 capítulos. También se adjuntan los anexos que contienen los diferentes artefactos que se generaron. Los contenidos principales de cada capítulo son los siguientes:

Capítulo 1: Fundamentación Teórica

Este capítulo establece la base teórica sobre la que se fundamenta la investigación, se plantean los conceptos básicos asociados a la Gestión de Riesgos de Seguridad Informática. Contiene un estudio y análisis crítico de las principales metodologías, técnicas y software utilizados en la Gestión de Riesgos de Seguridad Informática. Se definen las herramientas, lenguajes y metodologías que serán utilizadas para el desarrollo de la aplicación.

Capítulo 2: Propuesta de Solución

El capítulo 2 describe la propuesta de solución, la forma en que ésta pretende resolver el problema de la investigación. Se detalla el trabajo realizado en las primeras fases de la

metodología XP y los entregables generados como son las historias de usuarios, el Plan de iteraciones y el Plan de entrega.

Capítulo 3: Implementación y Pruebas

Este capítulo muestra el desarrollo del sistema a través de las 5 iteraciones planificadas, la arquitectura y los patrones de diseño aplicados. Se muestran las pruebas realizadas para garantizar la calidad del software mediante pruebas de aceptación y pruebas unitarias. También se presenta una evaluación de beneficios realizada por el cliente para comprobar en qué medida se cumplieron los objetivos trazados.

CAPITULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se exponen los conceptos principales asociados a la Gestión de Riesgos de Seguridad Informática. Se realiza un estudio y análisis crítico de las principales metodologías, técnicas y software utilizados en la Gestión de Riesgos de Seguridad Informática. Además se definen las herramientas, lenguajes y metodologías que serán utilizadas para el desarrollo de la aplicación.

1.2 Conceptos asociados

Para alcanzar un mejor entendimiento del campo de acción de la investigación es necesario profundizar en los principales conceptos asociados a la Gestión y Análisis de Riesgos de Seguridad Informática. A continuación se abordan las definiciones sobre las cuales se basa este trabajo.

1.2.1 Riesgo de Seguridad Informática

El riesgo es la estimación del grado de exposición a que una amenaza se materialice sobre uno o más activos de información, causando daños o perjuicios a la organización. El riesgo indica lo que le podría pasar a los activos si no se protegieran cabalmente. Para esto es importante saber qué características son de interés en cada activo, así como saber en qué medida estas características están en peligro (6).

1.2.2 Análisis de riesgos

Es un proceso sistemático para evaluar la magnitud de los riesgos a que se encuentra expuesta una organización. Una vez que se tiene la medida de los posibles eventos se puede definir qué hacer con los riesgos (6).

1.2.3 Gestión de riesgos

La gestión de riesgos es el proceso de selección e implantación de salvaguardas para, prevenir, impedir, reducir o controlar los riesgos identificados (6).

1.2.4 Gestión de riesgos en el Sistema de Gestión de Seguridad de la Información

Un Sistema de Gestión de Seguridad de la Información (SGSI) consiste en un conjunto de políticas, procedimientos, actividades y recursos asociados, gestionados colectivamente por una organización con el objetivo de proteger los activos de información. Un SGSI es un enfoque sistemático basado en procesos, para establecer, implementar, operar, monitorizar, revisar, mantener y mejorar la seguridad de la información de una organización. El SGSI se basa en la evaluación de riesgos y los niveles de aceptación de riesgo de la organización diseñados para tratar y administrar efectivamente los riesgos (7). La Gestión de Riesgos permite determinar cómo es, cuánto vale y cómo de protegido se encuentra el sistema para luego, elaborar un plan de seguridad que, implantado y operado, satisfaga los objetivos propuestos con el nivel de riesgo que acepta la Dirección (8).

1.3 Principales estándares de seguridad y metodologías para la Gestión de Riesgos

En el ámbito de la Seguridad Informática existen diferentes estándares, regulaciones, metodologías y marcos de trabajo que constituyen normas certificables y gozan de gran aceptación a nivel internacional. A continuación se detallan, a criterio del autor, las más importantes.

1.3.1 ISO/IEC 27000

La Organización Internacional para la Estandarización (ISO) y la Comisión Electrotécnica Internacional (IEC) han publicado un conjunto de estándares que constituyen una referencia a nivel internacional en la temática de seguridad de la información. La ISO ha dedicado la serie de numeración 27000 para las normas relacionadas con sistemas de gestión de Seguridad de la Información. Entre las que se encuentran:

1. 27000 (términos y definiciones).
2. 27002 (objetivos de control y controles, contiene 39 objetivos de control y 133 controles, agrupados en 11 dominios).
3. 27003 (guía de implantación de un SGSI).
4. 27004 (métricas y técnicas de medida de la efectividad de un SGSI).
5. 27005 (guía para la gestión del riesgo de seguridad de la información).

6. 27006 (proceso de acreditación de entidades de certificación y el registro de SGSI) (2).

1.3.2 ISO/IEC 27001

La ISO/IEC 27001 se le aplica a organizaciones que manejen información y que requieran protegerla, como es la impresa o escrita en papel, almacenada electrónicamente, enviada por correo u otro medio electrónico. Esta norma plantea la utilización del modelo PDCA (Plan - planificar, Do - hacer, Check –verificar, Act - actuar) que aporta a la organización importantes beneficios:

1. Establecimiento de una metodología de gestión de la seguridad clara y estructurada.
2. Reducción del riesgo de pérdida, robo o corrupción de información.
3. Los riesgos y sus controles son continuamente revisados.
4. Confianza de clientes y socios estratégicos por la garantía de calidad y confidencialidad comercial.
5. Las auditorías externas ayudan cíclicamente a identificar las debilidades del sistema y las áreas a mejorar.
6. Posibilidad de integrarse con otros sistemas de gestión (ISO 9001, ISO 14001, OHSAS 18001).
7. Continuidad de las operaciones necesarias de negocio tras incidentes de gravedad.
8. Conformidad con la legislación vigente sobre información personal, propiedad intelectual y otras.
9. Imagen de la empresa a nivel internacional y elemento diferenciador de la competencia.
10. Confianza y reglas claras para las personas de la organización.
11. Reducción de costes y mejora de los procesos y servicio.
12. Aumento de la motivación y satisfacción del personal.
13. Aumento de la seguridad en base a la gestión de procesos en vez de en la compra sistemática de productos y tecnologías (9).

Los estándares de gestión de la Seguridad de la Información son complementados con otros que tratan el tema de la gestión de riesgos. Entre los más aceptados se encuentran OCTAVE, MAGERIT, ISO/IEC 27005 y NIST SP 800-30.

1.3.3 ISO/IEC 27005

ISO/IEC 27005 proporciona directrices para la gestión de riesgos de seguridad de la información en una organización, específicamente apoyándose en los requerimientos establecidos por el SGSI definido por la ISO 27001. El estándar ISO 27005 es aplicable a todo tipo de organización, no provee o recomienda una metodología específica sino que describe a través de su clausulado el proceso recomendado de análisis incluyendo las fases que lo conforman:

1. Establecimiento del contexto.
2. Evaluación del riesgo.
3. Tratamiento del riesgo.
4. Aceptación del riesgo.
5. Comunicación del riesgo.
6. Monitorización y revisión del riesgo (10).

1.3.5 NIST SP 800-30

Las publicaciones especiales (SP) en su serie 800, de la división de seguridad del Instituto Nacional de Estándares y Tecnologías de EEUU (NIST) definen especificaciones técnicas, procedimientos y estándares sobre seguridad de la información. Esta serie en su publicación 30 presenta una metodología para el análisis y gestión de riesgos, que es alineada y complementaria con el resto de documentos de la serie. Esta metodología está basada en las directrices definidas en la ISO/IEC 27005. La Metodología NIST SP 800-30 está compuesta por 9 pasos básicos para el análisis de riesgo:

1. Caracterización del sistema de información.
2. Identificación de amenaza.
3. Identificación de vulnerabilidades.

4. Control de análisis.

6. Análisis de impacto.

7. Determinación del riesgo.

8. Recomendaciones de control.

9. Resultado de la implementación o documentación (10).

1.3.6 OCTAVE

La metodología de Evaluación de Vulnerabilidades y Amenazas Criticas Operacionales de Activos en inglés Operationally Critical Threats Assets and Vulnerability Evaluation (OCTAVE) se enfoca en dos aspectos diferentes: riesgos operativos y prácticas de seguridad. La tecnología es examinada en relación a las prácticas de seguridad, permitiendo a las compañías tomar decisiones de protección de información basadas en los riesgos de confidencialidad, integridad y disponibilidad de los bienes relacionados a la información crítica. El método OCTAVE permite la comprensión del manejo de los recursos, identificación y evaluación de riesgos que afectan la seguridad dentro de una organización. Exige llevar la evaluación de la organización y del personal de la tecnología de la información por parte del equipo de análisis mediante el apoyo de un patrocinador interesado en la seguridad. El método OCTAVE se enfoca en tres fases para examinar los problemas organizacionales y tecnológicos:

1. Identificación de la información a nivel gerencial.

2. Identificación de la información a nivel operacional.

3. Identificación de la información a nivel de usuario final (11).

1.3.4 Selección de la metodología MAGERIT

El Consejo Superior de Administración Electrónica (CSAE) desarrolla y promueve la Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información MAGERIT como respuesta a la percepción de que la administración de una organización depende de forma creciente de las tecnologías de la información para cumplir sus objetivos. MAGERIT está estrechamente relacionada con la generalización del uso de las TIC, que supone evidentes

beneficios para las organizaciones; pero también da lugar a ciertos riesgos que deben minimizarse con medidas de seguridad, en aras de infundir confianza en el uso de tales medios. MAGERIT persigue los siguientes objetivos:

1. Hacer conscientes a los responsables de los sistemas de información de la existencia de riesgos y de la necesidad de atajarlos a tiempo.

2. Proveer un método sistemático para analizar tales riesgos.

3. Ayudar a descubrir y planificar las medidas pertinentes para mantener los riesgos bajo control.

4. Preparar a la Organización para procesos de evaluación, auditoría, certificación o acreditación, según corresponda en cada caso (6).

Esta metodología presenta un amplio marco de trabajo definido en tres libros entre los que se encuentra un catálogo de elementos que incluye clasificaciones para los activos, amenazas y salvaguardas, una guía de técnicas que contiene diferentes formas de realizar el análisis de Riesgos por lo que se ajusta a los requerimientos planteados por la Dirección de Tecnología.

1.4 Análisis de soluciones similares

A continuación se presentan las características de dos de las herramientas con más reconocimiento en el Análisis y Gestión de Riesgos de Seguridad Informática.

1.4.1 R-Box

R-box es una solución modular para la Gestión de Seguridad de la Información y cumplimiento de estándares, desarrollado en Argentina por Khu Technologies S.A. En R-Box se han modelado todos los controles definidos en la Comunicación A 5374 del Banco Central de la

República Argentina (BCRA) donde se plantean requisitos mínimos de gestión y control de los riesgos relacionados con tecnología informática y sistemas de información (11).

Ventajas:

- Amplia base de conocimiento de entidades, controles, estándares y modelos.
- Gran flexibilidad de configuración y definición de parámetros.
- Sus funcionalidades permiten implantar metodologías coherentes y efectivas.
- Mejora continua de la herramienta (11).

Desventajas:

- Es una herramienta privativa.
- Está basado en la Comunicación A 5374 del BRCA que no es un estándar internacional y no es certificable.
- La información sobre todos los activos debe ser introducida por el usuario.

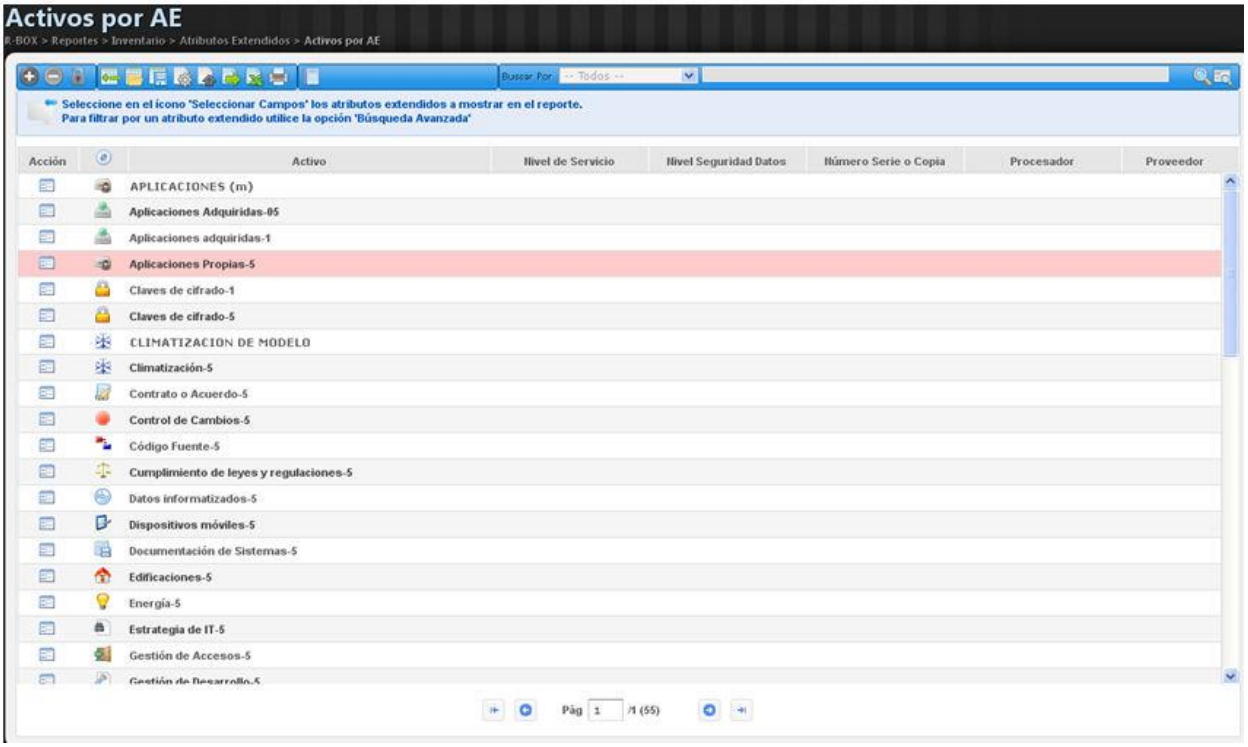

Figura 1: Informe de inventario de activos de R-Box

1.4.2 PILAR

PILAR es una herramienta capaz de realizar el análisis y la gestión de riesgos de un sistema de información siguiendo la metodología MAGERIT.

Ventajas

PILAR dispone además de una biblioteca estándar de propósito general, y es capaz de realizar calificaciones de seguridad respecto a normas ampliamente conocidas como son:

- ISO/IEC 27002 (2005, 2013)- Código de buenas prácticas para la Gestión de la Seguridad de la Información.
- ENS - Esquema Nacional de Seguridad (12).

Desventajas

- Es una herramienta privativa.
- No permite la identificación de activos automáticamente.

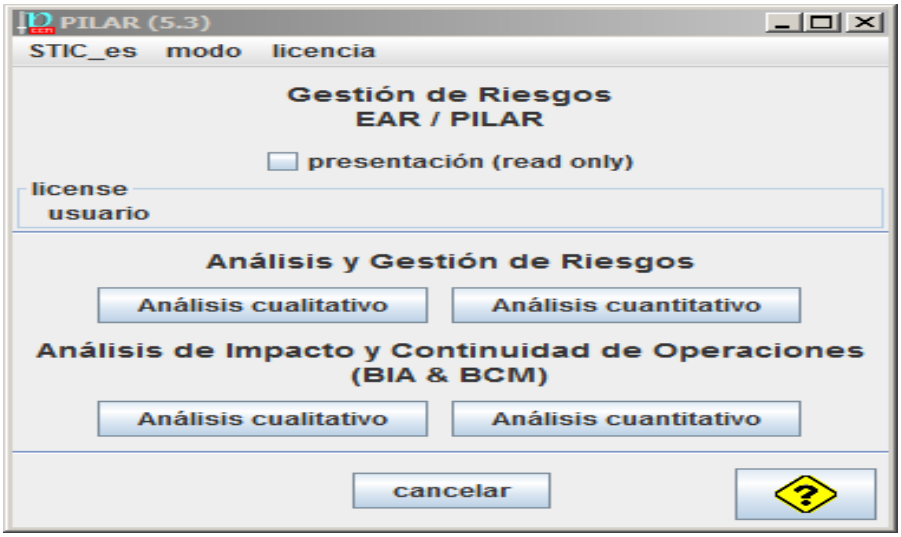

Figura 2: Entorno de Análisis de Riesgos PILAR.

1.4.3 Resultados del análisis de soluciones

Una vez presentadas las características de estas herramientas se pudo identificar sus principales desventajas, evidenciándose que no cumplen con las necesidades del cliente y no pueden ser adaptadas para resolver el problema de la investigación. No obstante, estas soluciones presentan algunas funcionalidades que podrían servir de guía a la solución propuesta como son los reportes gráficos y la apariencia intuitiva de R-Box y la biblioteca estándar para realizar clasificaciones de seguridad informática de la herramienta PILAR.

1.5 Metodologías de desarrollo de software

Una metodología de desarrollo de software es un conjunto de directivas, y documentación adjunta que guían el proceso de desarrollo, permitiendo su planificación, gestión, control y evaluación (13) (14) . Existen diversas metodologías de desarrollo divididas en dos enfoques principales: ágiles y tradicionales, destacándose las metodologías que se describen a continuación.

1.5.1 Extreme Programing (XP)

Extreme Programing (XP) es una metodología de desarrollo de software concebida y desarrollada para hacer frente a las necesidades específicas de desarrollo en las que equipos pequeños de trabajo se enfrentan a requisitos de software imprecisos y cambiantes. XP se caracteriza por la realimentación continua entre el cliente y el equipo de desarrollo. Esta metodología ágil desafía muchos principios convencionales, incluyendo la vieja certidumbre o suposición de que el costo de cambiar una pieza de software aumenta necesaria y dramáticamente en el transcurso de tiempo. XP reconoce que los proyectos tienen que trabajar para lograr esta reducción en el costo y explotar los ahorros una vez que se han ganado. Para lograrlo XP define los siguientes fundamentos:

1. Distinguir entre las decisiones a tomar por intereses comerciales y las que se hicieron por los interesados del proyecto.
2. Las pruebas unitarias se escriben antes de la programación y se mantiene la ejecución de pruebas en todo momento.
3. Integración y pruebas de todo el sistema varias veces al día.
4. La producción de todo el software se debe hacer en parejas.

5. Empezar los proyectos con un diseño simple que constantemente evoluciona para agregar flexibilidad y eliminar la complejidad innecesaria.
6. Poner un sistema mínimo en producción que rápidamente va creciendo en la dirección que demuestre ser más importante (15).

1.5.2 Scrum

Scrum proporciona un pequeño conjunto de reglas que crean suficiente estructura para que los equipos puedan centrar su innovación en la solución. La construcción de los productos complejos a los clientes es una tarea intrínsecamente difícil. Scrum proporciona un marco de trabajo para permitir a los equipos hacer frente a esa dificultad. Sin embargo, el proceso fundamental es simple y se rige por 3 funciones principales.

1. Los dueños del producto determinan que se va a construir en los próximos 30 días o menos.
2. Los equipos de desarrollo deben construir lo que se necesita en 30 días (o menos), y luego entregar lo que han construido. Sobre la base de esta entrega, el propietario del producto determina qué construir para la próxima entrega.
3. Los jefes de Scrum garantizan que este proceso ocurra en las mejores condiciones, y ayudan a mejorar continuamente el proceso, el equipo y el producto que se ha creado.

Si bien esta es una visión muy simplificada de cómo funciona Scrum, captura la esencia de este enfoque altamente productivo para la colaboración en equipo y desarrollo de productos de software (16).

1.5.3 Rational Unified Process (RUP)

RUP es un proceso de desarrollo de software que presenta tres características esenciales: está dirigido por casos de uso, está centrado en la arquitectura y es iterativo e incremental. RUP es concebido como un proceso de ingeniería de software y un marco genérico que puede especializarse para una gran variedad de sistemas de software, distintas áreas de aplicación y tipos de organizaciones. Su objetivo principal es asegurar la producción de software de alta calidad que cumpla con las necesidades de los usuarios, con una planeación y presupuesto predecible (17).

Tabla 1: Comparación entre las metodologías

Características	XP	RUP	SCRUM
Costo-cambios	Es capaz de reducir el costo asociado a un cambio en las etapas de vida del sistema.	Un cambio en las etapas de vida del sistema incrementaría notablemente el costo.	Reduce los costos del plan interno de mejora de procesos.
Trabajadores	Se requiere un grupo pequeño de programadores. Los programadores deben trabajar en pares.	Se requiere un equipo de trabajo de gran envergadura.	Necesita un equipo pequeño de desarrolladores desde 1-9 para trabajar con esta metodología.
Obtención de requisitos	Historia de usuario	Casos de uso	Resultados tangibles
Documentación	Poca	Mucha	Poca
Duración de Proyecto	Corta	Larga	Corto
Reutilización del código	Si	Si	Si
Simplicidad en el diseño	Si	No	Si
Diseño simple	Si	No	Si
Evaluación del estado de proyecto	Corto	Largo	Corto
Accesibilidad al código fuente por parte del cliente	Mucha	Poca	Mucha

1.5.4 Resultados del análisis de las metodologías de desarrollo de software

El proceso de desarrollo de software debe ser guiado por una metodología de desarrollo adecuada a las características del proyecto. En la presente investigación se opta por el uso de la metodología de desarrollo XP. Esta metodología de desarrollo de software es una de las más

exitosas en la actualidad, ideal para proyectos de corto alcance y pequeños equipos donde el cliente mantiene estrecha comunicación con el equipo de desarrollo.

1.6 Herramientas y lenguajes de programación

1.6.1 Herramienta de modelado Visual Paradigm

Visual Paradigm es una herramienta CASE (Ingeniería de Software Asistida por Computadora) que ayuda a los equipos de desarrollo de software en la realización de modelos de software UML. Permite documentar especificaciones de software y la generación de código fuente a partir del modelo de implementación. Permite realizar el análisis y diseño de todo tipo de diagramas UML. Visual Paradigm Standard Edition también es compatible con la programación de bases de datos mediante la generación de bases de datos y las clases ORM necesarias para acceder a la base de datos en el enfoque orientado a objetos, a través de Hibernate. Visual Paradigm Standard soporta ingeniería inversa de código a modelos, y de código a diagrama (18).

1.6.2 Lenguaje de Modelado Unificado

UML son las siglas de Unified Modeling Language (Lenguaje Unificado de Modelado), es un lenguaje que permite especificar, visualizar y construir artefactos necesarios para el desarrollo de sistemas de software. Es un sistema de notación (que incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos. Independientemente de la aceptación que ha tenido, este lenguaje ha recibido la aprobación de facto en la industria de software (19).

1.7 Tecnología del lado del servidor.

1.7.1 Lenguaje PHP

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto esencialmente utilizado para el desarrollo web y que puede ser embebido en código HTML. El código PHP es ejecutado en el servidor, generando HTML y enviándolo al cliente (20). Una de las facilidades de este lenguaje es la gran diversidad de bases de datos que soporta como son: Oracle, PostgreSQL y MySQL. PHP es capaz de utilizar servicios que usen protocolos como IMAP, SNMP, NNTP, POP3, HTTP y sus derivados y habilitar sockets de red directos (raw

sockets). Los scripts PHP también se pueden ejecutar sin necesidad de servidor web o navegador. Solamente se necesita el intérprete PHP para usarlo de esta manera. Este tipo de uso es ideal para scripts ejecutados regularmente desde cron (Linux) o el Planificador de tareas en Windows (20).

1.7.2 Servidor web Apache

Un servidor web es una aplicación que continuamente atiende peticiones de los navegadores, proporcionando los recursos que estos soliciten usando el protocolo HTTP o HTTPS. En general un servidor web se mantiene activo realizando las siguientes acciones:

1. Escucha peticiones en el puerto TCP indicado (el estándar por defecto para HTTP es el 80).
2. Recibe una petición.
3. Busca el recurso.
4. Envía el recurso utilizando la misma conexión que recibió petición.
5. Regresa al segundo paso.

Apache es un servidor web gratuito de código abierto, altamente configurable y de diseño modular. Apache se levanta como un servicio del sistema y convierte la máquina en un servidor capaz de enviar contenido a cualquier navegador. Para comprobar si el servidor funciona se debe ir al localhost y un mensaje informará si el servidor está prestando su servicio. A partir de ese momento se debe copiar el contenido web que se quiera servir al directorio www. Existe una gran cantidad de módulos Apache disponibles para su utilización. Soporta Perl, PHP y otros lenguajes script. Apache funciona en Linux y en otros sistemas de Unix, así como en Windows (21).

1.8 Tecnologías del lado del cliente

Las tecnologías del lado del cliente son un conjunto de lenguajes y técnicas que pueden ser interpretadas por el navegador de forma directa sin necesidad de tratamiento previo. En este epígrafe se describen las tecnologías del lado del cliente a utilizar en el desarrollo del sistema.

1.8.1 Hyper Text Markup Language. HTML5

El Lenguaje de Marcado de Hipertexto es un lenguaje de composición de documentos y especificación de hipervínculos (link) de hipertexto que define la sintaxis y coloca instrucciones especiales transparentes al usuario (22). La sintaxis y semántica básica de HTML está definida

en el estándar de HTML, actualmente en su versión 5. HTML5 es considerado un nuevo concepto para la construcción de sitios web y aplicaciones en una era que combina dispositivos móviles, computación en la nube y trabajos en red. HTML5 provee básicamente tres características: estructura, estilo y funcionalidad. Se puede entender como el producto de la combinación de HTML, CSS y Javascript. Estas tecnologías están altamente relacionadas y actúan como una sola unidad organizada bajo la especificación de HTML5. HTML está a cargo de la estructura, CSS presenta esa estructura y su contenido en la pantalla y Javascript hace el resto (23).

1.8.2 Javascript

Javascript es una implementación que realizó la empresa Netscape del estándar ECMAScript. Es un lenguaje de programación del lado del cliente, que se utiliza principalmente para crear páginas web dinámicas. Debido a que presenta gran compatibilidad con la mayoría de los navegadores actuales, se ha convertido en el lenguaje de programación del lado del cliente más utilizado. Con Javascript se pueden crear efectos en las páginas y definir interactividades con el usuario. Una de sus ventajas más importantes consiste en la reducción de carga del servidor, Javascript se hace cargo de gran parte de las funciones del cliente de las cuales se encargaba anteriormente el servidor, como la validación de campos (24) (25).

1.8.3 Cascading Style Sheets (CSS)

CSS es el lenguaje utilizado para definir el estilo o aspecto de un documento HTML. Con CSS se puede separar la definición de los contenidos y la definición de su aspecto. CSS conserva el ancho de banda del usuario lo que acelera la carga de las páginas. Los estilos son aplicados a varios elementos, lo que permite lograr uniformidad y visualizar el mismo documento en varios dispositivos diferentes, así como reducir el tiempo de diseño y programación de las vistas de los sitios web (26). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y la actualización ya que los estilos se encuentran en unos pocos archivos CSS, al editarlos los cambios se realizarán en la parte del documento HTML que actúen (27).

1.9 Entorno de desarrollo integrado (IDE) NetBeans.

Un entorno de desarrollo integrado (IDE) es un software generalmente compuesto por un editor de código fuente, un compilador y un intérprete, la automatización de generación de herramientas y un depurador. El objetivo principal de un IDE es prestar servicios integrales a

los programadores de software. Para el desarrollo de la propuesta de solución se utilizará el IDE NetBeans que presenta las siguientes funcionalidades:

- Desarrollo general de Java, incluyendo conceptos básicos de configuración del proyecto, construcción, pruebas y depuración de aplicaciones Java.
- Integración con herramientas y servicios externos. Incluyendo la integración con bases de datos y el uso de sistemas de gestión de código fuente.
- PHP y HTML5. Incluyendo la edición de PHP, JavaScript, CSS y depuración de aplicaciones.
- Aplicaciones móviles y embebidas.
- Integración con Symfony2 (28).

1.10 Framework de desarrollo

1.10.1 Symfony 2.3.5

Symfony es framework de desarrollo de web que automatiza tareas comunes como la creación de entidades, el esquema de la base de datos y los formularios. Esto libera en gran medida al desarrollador, el cual puede dedicarse por completo a los aspectos específicos de cada aplicación. Symfony implementa el patrón de diseño Modelo-Vista-Controlador, separando la lógica del negocio, la lógica del servidor y la presentación de la aplicación web. Symfony2 ha sido ideado para explotar al máximo todas las nuevas características de PHP 5 siendo uno de los frameworks PHP con mejor rendimiento. Este framework es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft, siendo independiente del sistema de gestión de base de datos. Se puede ejecutar tanto en plataformas Unix, Linux como en plataformas Windows. Symfony 2.3 se publicó en junio de 2013, es una versión que mejora muchos aspectos de la versión original, al tiempo que mantiene una alta compatibilidad con versiones anteriores (29).

1.10.2 Bootstrap 3

Bootstrap es un framework de desarrollo web para el lado del cliente. Fue creado por desarrolladores de Twitter y se ha convertido en uno de los más utilizados para la creación de interfaces web con CSS y Javascript. Los diseños de las interfaces de Bootstrap son sencillos e intuitivos, esto trae como consecuencia una mayor ligereza y rapidez a la hora de cargar y al adaptarse a otros dispositivos. Además de esto, las interfaces desarrolladas con Bootstrap

presentan un diseño adaptativo ya que son capaces de reajustar la interfaz dependiendo del tamaño del dispositivo en el que se visualice. El framework contiene elementos muy comunes a la hora de desarrollar interfaces web como son los botones, menús desplegables y formularios predefinidos con estilos y que son fáciles de configurar y personalizar (30) (31).

1.11 Sistema gestor de bases de datos MySQL

MySQL es un sistema de administración de bases de datos relacional de código abierto que goza de gran aceptación internacional. Es un software capaz de almacenar considerables cantidades de datos y distribuirlos para cubrir las necesidades de una institución de cualquier tipo, desde establecimientos comerciales a grandes empresas y organismos administrativos. MySQL permite establecer diferentes niveles de acceso de usuario, administrar el sistema, proteger y hacer volcados de datos. Además permite desarrollar aplicaciones de base de datos en la mayor parte de los lenguajes de programación utilizados en la actualidad y ejecutarlos en casi todos los sistemas operativos. Se pueden efectuar la optimización, análisis, comprobación y reparación de tablas. MySQL es muy efectivo al realizar copias de seguridad y duplicados de la base de datos, es capaz de crear índices y optimizar consultas. Es un gestor de bases de datos muy fácil de configurar y optimizar, siendo sus archivos de configuración una forma flexible de controlar el comportamiento del servidor (32). Este gestor de bases de datos es utilizado por las demás herramientas ya desplegadas por la Dirección de tecnología de la Facultad 4 por lo que será utilizado para solución propuesta.

1.12 Propuesta y selección de las herramientas

Para desarrollar un sistema que se ajuste a las necesidades identificadas por el Departamento de Tecnología de la Facultad 4 se selecciona la metodología ágil XP ya que concuerda con las características del proyecto a realizar y del equipo de trabajo. Para la implementación del sistema se utilizará como tecnologías del lado del cliente JavaScript, CSS, HTML. Como tecnologías del lado del servidor se utilizará como lenguaje PHP y Apache como servidor web. Como frameworks de desarrollo web se utilizará Bootstrap 3 para CSS y para PHP se utilizará Symfony en su versión 2.3.5. Para gestionar la base de datos del sistema se utilizará el gestor de bases de datos MySQL por su fiabilidad y capacidad, y con el objetivo de lograr completa integración con los demás sistemas implantados por el cliente que utilizan este gestor de bases de datos.

1.13 Conclusiones del capítulo

Mediante el conocimiento de los principales conceptos asociados a la Gestión y Análisis de Riesgos de Seguridad Informática se ha constatado la importancia y el papel que cumple, dentro de la Gestión de la Seguridad Informática, la Gestión de Riesgos, por cuanto permite evaluar de manera acertada los eventos que pueden provocar un impacto negativo en la organización y sienta las bases para tomar decisiones que permitan proteger el negocio.

Luego de un análisis de las diferentes metodologías que guían el proceso de Gestión de Riesgos de Seguridad Informática, se elige la metodología MAGERIT como guía para el desarrollo de la solución, por la claridad y alcance de su propuesta. Se debe señalar que la solución propuesta no es una implementación exacta de la metodología MAGERIT, pues estará adecuada al entorno tecnológico y organizacional que es objeto de investigación.

Durante el estudio del arte se evidenció la escasez de herramientas libres para el análisis y gestión de riesgos. Al realizarse una descripción de las aplicaciones R-BOX y PILAR, que se utilizan como herramientas para gestión de riesgos, se evidenció que no cubren todas las necesidades planteadas por el cliente, ante todo por el hecho de ser herramientas privativas. Además de esto, se evidenció la necesidad de contar con una herramienta capaz de desempeñar automáticamente ciertas tareas de menor importancia pero si costosas en tiempo como la identificación y valoración de activos del sistema de información.

Las herramientas y tecnologías seleccionadas se corresponden de forma general con las características del sistema a implementar. Estas herramientas y tecnologías contribuirán a optimizar el desarrollo del sistema y el desempeño de sus funcionalidades.

CAPÍTULO 2 PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se describirá de forma detallada la propuesta de solución y como esta pretende resolver el problema de la investigación. Se realizará la descripción del trabajo en las primeras fases de la metodología XP y los entregables generados como son las historias de usuarios, el Plan de iteraciones y el Plan de entrega.

2.2 Introducción a MAGERIT-versión 3.0

Como se especificó en el capítulo anterior se decide utilizar la metodología MAGERIT como guía para la realización del sistema. MAGERIT implementa el Proceso de Gestión de Riesgos dentro de un marco de trabajo para que los órganos de gobierno tomen sus decisiones teniendo en cuenta los riesgos derivados del uso de tecnologías de la información (8). La versión 3 de MAGERIT ha sido estructurada (como la versión 2) en dos libros y una guía de técnicas: el Libro I— Método, describe cómo hacer la Gestión de Riesgos. El Libro II – Catálogo de elementos, brinda un ítem estándar a los que puedan acogerse, aquellos que realizan el método, de forma rápida, centrándose en lo específico del sistema objeto del análisis. La guía de Técnicas contiene una recopilación de técnicas de diferentes tipos que pueden ser de utilidad para la aplicación del método. MAGERIT divide el Proceso de Gestión de Riesgos en dos subprocesos: el Análisis y la Gestión. El análisis de riesgos es una aproximación metódica para determinar el riesgo cumpliendo los siguientes pasos:

1. Determinar los activos relevantes para la Organización, su interrelación y su valor, en el sentido de qué perjuicio supondría su degradación.
2. Determinar a qué amenazas están expuestos aquellos activos.
3. Determinar las salvaguardas 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 por la tasa de ocurrencia (o expectativa de materialización) de la amenaza.

La Gestión de Riesgos toma los datos aportados por el Análisis: los impactos y riesgos a que está expuesto el sistema, para tomar una serie de decisiones condicionadas por diversos factores como la gravedad del impacto y/o del riesgo, las obligaciones a las que por ley esté sometida la organización, las obligaciones a las que por reglamentos sectoriales esté sometida la organización o las obligaciones a las que por contrato esté sometida. Los pasos a realizar para la Gestión de Riesgos son:

1. Evaluación: interpretación de los valores de impacto y riesgo residuales.
2. Tratamiento: la Dirección puede decidir si aplicar algún tratamiento al sistema de seguridad desplegado para proteger el sistema de información (8).

Los procesos anteriormente descritos constituyen el núcleo conceptual y funcional del sistema a desarrollar.

2.3 Propuesta de solución

En pos de cumplir el objetivo general de la presente investigación, y en respuesta a las especificaciones presentadas por la Dirección de Tecnología de la Facultad 4 se propone el desarrollo del Sistema Informático para el Análisis y Gestión de Riesgos de Seguridad Informática en la Facultad 4 (SAGRSI) que constituirá el principal aporte del presente trabajo. El objetivo principal de este sistema es servir como herramienta para la realización del proceso de Gestión de Riesgos de Seguridad Informática, liberando al usuario de tareas como la identificación de activos no esenciales y amenazas, facilitando la valoración y clasificación de los activos mediante un catálogo de elementos abarcador para homogenizar los resultados de estas tareas. El SAGRSI estará integrado a GLPI (Gestionnaire Libre de Parc Informatiqué). GLPI es una solución Open Source para la gestión del inventario informático y de soporte técnico (Help Desk). Es una aplicación Web que ataca los principales problemas de gestión del inventario informático: la administración de los recursos de hardware, software, usuarios, suministros e incidencias (33). SAGRSI tomará los datos sobre los activos de información no esenciales (pero necesarios para el análisis de riesgo), del inventario de GLPI automáticamente. De igual forma, SAGRSI hará uso de las amenazas reportadas en las incidencias de GLPI, permitiendo así la identificación y cálculo automático de la frecuencia de ocurrencia de éstas. SAGRSI incluye gran parte del Catálogo de Elementos definido por la MAGERIT para la clasificación de los activos, amenazas y salvaguardas. El sistema realizará el

Análisis de Riesgos Algorítmico por el Método Cuantitativo descrito en la Guía de Técnicas y Herramientas de la siguiente forma:

2.3.1 El valor de los activos

El valor de un activo en cierta dimensión (integridad, accesibilidad, disponibilidad, etc.) es un valor real superior a cero.

2.3.2 Jerarquía y dependencias entre activos

Interesa tanto saber si un activo A depende o no de otro activo B, como saber en qué grado depende. Esta visión concibe el sistema como un grafo ponderado o un árbol ya que existe jerarquía de activos:

1. Activos esenciales
    - Información que se maneja
    - Servicios prestados
2. Servicios internos que estructuran ordenadamente el sistema de información
3. El equipamiento informático
    - Aplicaciones(software)
    - Equipos informáticos (hardware)
    - Comunicaciones
    - Soportes de información: discos, cintas, etc.
4. El entorno: activos que se precisan para garantizar las siguientes capas
    - Equipamiento y suministros: energía, climatización, etc.
    - Mobiliario
5. Las instalaciones físicas.

El grado de dependencia es un valor entre 0 y 1. La dependencia puede ser directa o indirecta, esta última se calculará por el cierre transitivo de las dependencias directas entre activos:

$$A \Rightarrow C \Leftrightarrow \exists\, B,\ (A \Rightarrow B) \wedge (B \rightarrow C)$$

Calculando el grado de dependencia como:

$$\text{grado}(A \Rightarrow C) = \sum_i \{ \text{grado}(A \Rightarrow B_i) \times \text{grado}(B_i \rightarrow C) \}$$

Donde las sumas se realizan de acuerdo con esta fórmula**:**

$$a + b = 1 - (1 - a) \times (1 - b)$$

2.3.3 El valor e impacto acumulado

Sea SUP (B) el conjunto de superiores de B, es decir el conjunto de activos que dependen directa o indirectamente B:

$$\text{SUP}(B) = \{ A_i,\ A_i \Rightarrow B \}$$

Se define el valor acumulado sobre B como la suma de valores de los activos superiores, ponderados por el grado de dependencia:

$$\text{valor\_acumulado}(B) = \text{valor}(B) + \sum_i \{ \text{valor}(A_i) \times \text{grado}(A_i \Rightarrow B) \}$$

El impacto acumulado de una amenaza sobre un activo es la pérdida de valor acumulado por la degradación que ésta provoca:

$$\text{impacto} = i = v \times d$$

2.3.4 Impacto repercutido

Si el activo A depende del activo B, las amenazas sobre B repercuten sobre A. El impacto se calcula como:

$$\text{impacto} = v \times d \times \text{grado}(A \Rightarrow B)$$

2.3.5 Paquete de salvaguardas

Frente a una amenaza se despliega un paquete de salvaguardas (PS). El paquete de salvaguardas es un conjunto de protecciones singulares acumuladas sobre un activo. La eficacia de un PS se calcula de la siguiente forma:

$$\min \{ 1{,}0, \sum_k e(ps_k) \}$$

El valor *e* es la eficacia de una salvaguarda y se puede descomponer en una eficacia frente al impacto ei y una eficacia frente a la frecuencia ef, de forma que:

$$(1 - e^i) \times (1 - e^f) = 1 - e$$

2.3.6 Riesgo

El riesgo se mide en las mismas unidades que el valor. El riesgo se calcula como:

$$\text{riesgo} = \text{impacto} \times \text{frecuencia}$$

Riesgo acumulado: En el cálculo del riesgo acumulado, se usará el impacto acumulado sobre el activo; es decir, la pérdida de valor acumulado por amenazas sobre el mismo.

Riesgo repercutido: En el cálculo del riesgo repercutido, se usará el impacto repercutido sobre el activo; es decir, la pérdida de valor propio por amenazas en activos inferiores.

Riesgo residual: Puede derivarse indirectamente como riesgo residual = impacto residual x frecuencia residual, estos últimos son resultados de aplicar una salvaguarda con cierta ei y ef.

2.4 Usuarios del sistema

La Gestión de Riesgo es un proceso vital en la Gestión de la Seguridad Informática, solo puede ser realizado por el personal que gobierna las TIC que tienen la responsabilidad y potestad para la toma las decisiones. En general el usuario de SAGRSI es el personal autorizado experto en tecnologías y sistemas de información o personal técnico cualificado del dominio afectado, el encargado de la gestión de seguridad informática y de la aplicación de la metodología de análisis y gestión de riesgos en particular.

Tabla 2: Roles del sistema

Rol	Descripción
Administrador	Es el único usuario del sistema que tiene acceso a todas sus funcionalidades. Efectúa la clasificación, evaluación y tratamiento de los riesgos. Es el responsable de la configuración del sistema y la gestión de usuarios.
Usuario	Es el encargado de valorar los activos de información en cada una de las dimensiones de Seguridad Informática que considere necesario. Realiza la clasificación de las amenazas y salvaguardas utilizando el catálogo de elementos. Introduce en el sistema las salvaguardas disponibles para proteger los activos. Genera informes.

2.5 Lista de funcionalidades del sistema

Para que el sistema pueda de satisfacer los objetivos propuestos, debe ser capaz realizar determinadas operaciones. A continuación se mencionan las funcionalidades a implementar:

1. Identificar los activos de información no esenciales en el inventario de GLPI.
2. Establecer dependencias entre los activos no esenciales del inventario de GLPI
3. Clasificar los activos de información no esenciales.
4. Gestionar activos de información.
5. Permitir al usuario valorar los activos de información.
6. Permitir al usuario establecer dependencias directas entre activos de información.
7. Establecer dependencias indirectas entre activos de información.

8. Calcular grado de dependencias indirectas entre activos.
9. Calcular valor acumulado del activo.
10. Generar informe Modelo del valor.
11. Identificar las amenazas reportadas en las incidencias de GLPI.
12. Gestionar amenazas.
13. Permitir al usuario relacionar las amenazas con los activos afectados.
14. Calcular impacto repercutido y acumulado de una amenaza.
15. Generar informe Mapa de Riesgos.
16. Gestionar salvaguarda.
17. Calcular eficacia de un paquete de salvaguardas sobre un activo.
18. Permitir al usuario aplicar una salvaguarda a un tipo de activo.
19. Generar informe Evaluación de salvaguardas
20. Estimar riesgo acumulado.
21. Estimar riesgo repercutido.
22. Estimar riesgo residual.
23. Estimar riesgo potencial.
24. Generar informe Estado de Riesgos.
25. Gestionar riesgos.
26. Gestionar usuarios.
27. Autenticar usuario.

2.6 Restricciones del sistema

Usabilidad: El sistema debe tener una interfaz intuitiva y sencilla. Debe representar claramente el orden cronológico de los procesos de gestión de riesgos y sus actividades. Debe permitir acceder directamente a elementos y actividades.

Seguridad: el acceso al sistema estará restringido por usuario y contraseña. Solo se permitirá al rol de Administrador gestionar los usuarios, asignar roles y configurar el sistema. Se deberá garantizar las tres características más importantes que protege la seguridad informática:

- **Accesibilidad:** Sólo podrán acceder al sistema, aquellas personas que se autentiquen correctamente en el mismo.
- **Integridad:** Los datos sólo podrán ser modificados por aquellas personas autorizadas para hacerlo.
- **Disponibilidad:** El sistema debe mantenerse funcionando permanentemente y garantizar que los datos estén disponibles para los usuarios autorizados.

Hardware: El sistema estará montado en una PC servidor junto con GLPI y debe cumplir con las siguientes características:

- 2GB de memoria RAM o superior.
- Procesador Dual-Core con frecuencia de reloj de 2.8 GHz o superior.
- 500 GB de almacenamiento interno o superior.

2.7 Fase de Exploración

Durante la fase de exploración los programadores comienzan a familiarizarse con las tecnologías que serán usadas para el desarrollo del sistema y se realizan pruebas para conocer los límites de desempeño de éstas. Mientras el equipo de desarrollo se encuentra explorando las tecnologías, el cliente comienza a entregar las historias de usuario que considera deben estar en la primera entrega del producto. En esta fase también se exploran las diferentes variantes de la arquitectura del sistema, varias parejas de programadores modelan el sistema de diferentes formas y luego se reúnen para comparar sus propuestas (15).

2.7.1 Historias de Usuarios.

Las Historias de Usuario (HU) son una breve descripción del comportamiento del sistema desde el punto de vista del usuario (15). Mediante esta técnica XP logra capturar los requisitos funcionales del sistema. Los autores de estas HU proponen una planilla que contiene los siguientes datos:

Nombre: nombre de la HU.
Número: número de la HU.
Usuario: el usuario del sistema o el protagonista de la HU.
Iteración: la iteración en que se va a implementar la HU.
Prioridad: es la importancia que tiene para el cliente y para el grupo de desarrollo (baja, media, alta).
Complejidad: es el nivel de dificultad que establece el desarrollador a la hora de implementar (baja, media, alta).
Punto estimado: es la estimación definida por el equipo de desarrollo del tiempo de duración para realizar la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo. En la metodología XP está definida una semana ideal como 5 días hábiles trabajando 40 horas, es decir, 8 horas diarias.
Descripción: es un pequeño resumen de la HU donde se describen las acciones del mismo.
Observación: es la nota de interés.

A continuación se muestran dos de las historias de usuarios entregadas por el cliente y que serán utilizadas más tarde en las pruebas de aceptación (Ver Anexo 1).

Tabla 3: HU- Identificar los activos del inventario de GLPI.

Historia de Usuario	
Nombre: Identificar los activos del inventario de GLPI	**Número:** 1
Usuario: Administrador	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1

Descripción: El sistema carga y muestra los activos de información no esenciales del inventario de GLPI con la clasificación según MAGERIT.
Observación:

Tabla 4: HU- Establecer dependencias entre los activos del inventario de GLPI

Historia de Usuario	
Nombre: Establecer dependencias entre los activos del inventario de GLPI	**Número:** 2
Usuario: Administrador	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1
Descripción: El sistema muestra los activos de información no esenciales del inventario de GLPI con las dependencias identificadas en el inventario de GLPI.	
Observación: El grado de dependencia tendrá valor por defecto 1.	

2.8 Fase de Planificación

El objetivo principal de la fase de planificación es que el cliente y el equipo de desarrollo lleguen a un acuerdo sobre la fecha en el que, el menor y más importante conjunto de historias serán implementadas (15). En esta fase se realiza una estimación del esfuerzo por HU y se define el orden en que serán implementadas.

2.8.1 Estimación del esfuerzo por HU

Se realizó una estimación del tiempo para cada HU identificada, a partir de la experiencia del desarrollador en otros proyectos similares. También se mantiene una retroalimentación con las historias ya realizadas para mejorar las estimaciones en las iteraciones siguientes. A continuación se muestran los resultados obtenidos.

Tabla 26: Estimación de esfuerzo por HU

Número	Historia de usuario	Puntos de estimación
1	Identificar los activos del inventario de GLPI.	1
2	Establecer dependencias entre los activos del inventario de GLPI.	1
3	Gestionar activos.	0.5
4	Valorar los activos.	0.5
5	Establecer dependencias directas entre activos.	0.5
6	Establecer dependencias indirectas de los activos.	1
7	Calcular valor acumulado del activo.	0.5
8	Generar informe Modelo del valor.	1
9	Identificar las amenazas desde GLPI.	0.5
10	Gestionar amenazas.	0.5
11	Relacionar las amenazas con los activos afectados.	0.5
12	Calcular impacto repercutido y acumulado de una amenaza.	0.5
13	Generar informe Mapa de Riesgos.	1
14	Gestionar salvaguarda.	0.5
15	Calcular eficacia de un paquete de salvaguardas.	0.5

16	Aplicar una salvaguarda a un activo de información.	0.5
17	Generar informe Evaluación de salvaguardas	0.5
18	Estimar riesgo acumulado.	0.2
19	Estimar riesgo repercutido.	0.2
20	Estimar riesgo residual	0.2
21	Estimar riesgo potencial	0.2
22	Generar informe Estado del Riesgo	1
23	Gestionar Riesgo	0.2
24	Autenticar Usuario	0.5
25	Gestionar usuario	0.5

2.8.2 Plan de Iteraciones

La metodología XP trabaja con iteraciones cortas donde el cliente selecciona las HU a realizar con el objetivo de maximizar el valor del negocio una vez terminada la iteración. El proyecto se desarrollará en cinco iteraciones según se describe en el Plan de Iteraciones.

Tabla 27: Plan de duración de las iteraciones

Iteraciones	Orden de las HU	Duración
Iteración 1	1. Identificar los activos del inventario de GLPI. 2. Establecer dependencias entre los activos del inventario de GLPI. 3. Gestionar activos.	3 semanas

	4. Valorar los activos.	
Iteración 2	5. Establecer dependencias directas entre activos. 6. Establecer dependencias indirectas de los activos. 7. Calcular valor acumulado del activo. 8. Generar informe Modelo del valor.	3 semanas
Iteración 3	9. Identificar las amenazas desde GLPI. 10. Gestionar amenazas. 11. Relacionar las amenazas con los activos afectados. 12. Calcular impacto acumulado y repercutido de una amenaza. 13. Generar informe Mapa de Riesgos.	3 semanas
Iteración 4	14. Gestionar salvaguarda. 15. Calcular eficacia de un paquete de salvaguardas. 16. Aplicar una salvaguarda a un activo de información. 17. Generar informe Evaluación de salvaguardas.	3 semanas
Iteración 5	18. Estimar riesgo acumulado.	

	19. Estimar riesgo repercutido. 20. Estimar riesgo residual. 21. Estimar riesgo potencial. 22. Generar informe Estado del Riesgo. 23. Gestionar Riesgo. 24. Gestionar Usuario 25. Autenticar Usuario.	3 semanas

2.8.3 Plan de entrega

XP plantea que las entregas deben ser pequeñas y frecuentes con el objetivo de garantizar rápidamente los beneficios esperados por el cliente y, al mismo tiempo, los programadores puedan retroalimentarse con más frecuencia sobre lo que el cliente desea. En la siguiente tabla se muestra el Plan de entrega.

Tabla 28: Plan de entrega

Historia de usuario	Primera iteración	Segunda iteración	Tercera iteración	Cuarta iteración	Quinta iteración
V 0.1	**20-02-2015**				
V 0.2		**13-03-2015**			
V 0.4			**2-04-2015**		
V 0.6				**24-04-2015**	
V 1.0					**15-05-2015**

2.9 Tarjetas Clase-Responsabilidad-Colaborador

Las tarjetas Clase-Responsabilidad-Colaborador (CRC) es una técnica de diseño orientado a objetos que describe las clases con sus responsabilidades y las clases con las que colaboran para realizar dichas responsabilidades. La tarjeta CRC es una herramienta con la cual se establecen las responsabilidades y se indica una colaboración con otros objetos. Fueron inventadas por Kent Beck y Ward Cunningham, para que los diseñadores de objetos piensen en términos más abstractos sobre la asignación de responsabilidades y de las colaboraciones. El objetivo de esta técnica es levantar un inventario de las clases del sistema y la forma en que interactúan entre ellas (19) (34).

Tabla 29: Tarjeta CRC ActivoController

Clase: ActivoController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a Activos. Listar activos.	Activo.

Tabla 30: Tarjeta CRC Activo

Clase: Activo	
Responsabilidades	**Colaboraciones**
Modelar el activo de información y su lógica de negocio. Obtener propiedades de un activo. Modificar propiedades de un activo.	ActivoSubtipo

Tabla 37: Tarjeta CRC Arboldeactivo

Clase: Arboldeactivo	
Responsabilidades	**Colaboraciones**
Modelar al sistema de información como un árbol de dependencias de activos de información. Calcular dependencias indirectas entre activos. Calcular valor acumulado de un activo. Calcular riesgo repercutido. Calcular riesgo acumulado	Nodo

Tabla 38: Tarjeta CRC Nodo

Clase: Nodo	
Responsabilidades	**Colaboraciones**
Modelar al activo como parte orgánica del sistema de información con sus dependencias (activos hijos y superiores), valoraciones, riesgos etc. Obtener todas las propiedades de un nodo del árbol de activos.	Activo ActivoActivo Valoración Riesgo Amenaza

2.10 Prototipo de interfaz de usuario

El prototipo de interfaz de usuario tiene como objetivo principal que el usuario conozca cómo se presentará la información en el sistema, para que luego exprese sus consideraciones y necesidades, estableciéndose la retroalimentación deseada entre este y el equipo de desarrollo. A continuación se muestra el prototipo de interfaz no funcional para el Inventario de activos y un prototipo para la forma en que se mostrará la información de un activo.

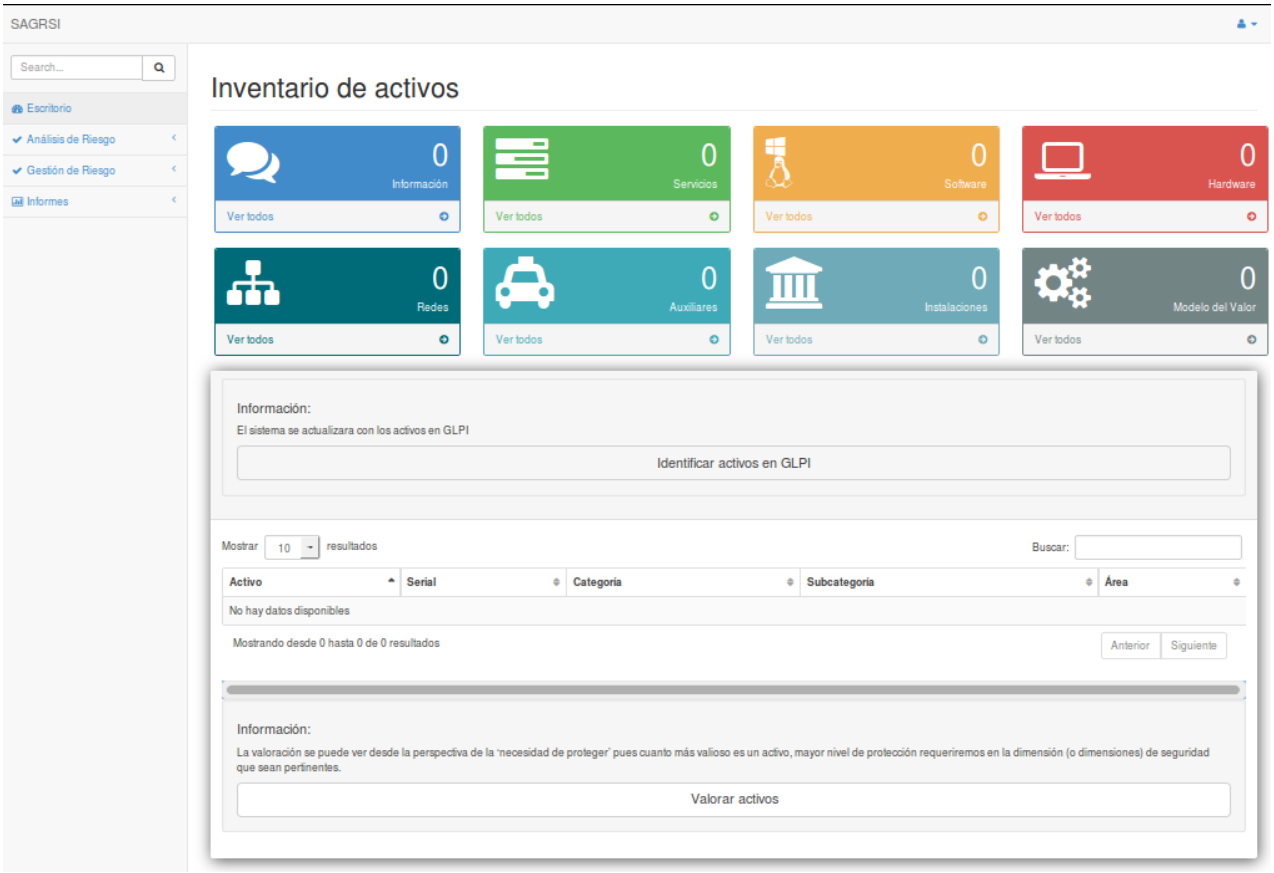

Figura 3: Inventario de activos

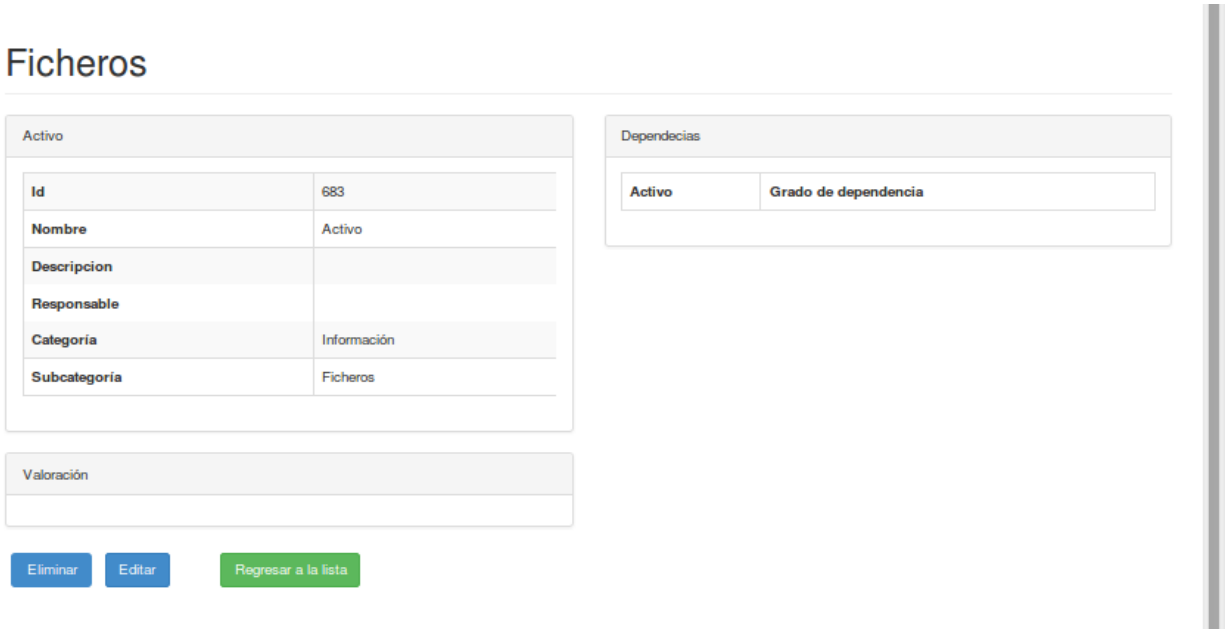

Figura 4: Mostrar Activo

2.11 Conclusiones del capítulo

La metodología MAGERIT aportó el marco conceptual y el enfoque de proceso continuo a que se acoge el sistema propuesto. La utilización de la base de datos de GLPI, la cual es actualizada automáticamente por el sistema OCS Inventory, le permite a SAGRSI conocer sobre el estado de la gran parte de los activos de información; esto sumado a la clasificación automática, la identificación de las amenazas y el cálculo de su frecuencia de ocurrencia, supone una disminución del tiempo en que se realizan las tareas de identificación de activos, la disminución del tiempo de adaptación a los cambios en el sistema de información y garantiza una mayor integridad de los datos (véase Capítulo 3).

Mediante la utilización de las historias de usuario de XP se logró obtener de forma sencilla y dinámica los requisitos funcionales del sistema. XP propició la retroalimentación con el cliente y la organización de las diferentes iteraciones. El uso de técnicas de diseño como las tarjetas CRC brindó una perspectiva sencilla de la arquitectura del sistema, contribuyendo a elevar la productividad del equipo de desarrollo y disminuyendo los riesgos de implementación de software.

CAPITULO 3. IMPLEMENTACIÓN Y PRUEBAS

3.1 Introducción

En el presente capítulo se describe el desarrollo del sistema a través de las 5 iteraciones planificadas, la arquitectura y los patrones de diseño utilizados. En cada iteración se realizó una entrega al cliente de una parte funcional del sistema, siendo la primera entrega la que materializa la arquitectura propuesta. Para garantizar la calidad del software se confeccionaron pruebas de aceptación y pruebas unitarias.

3.2 Arquitectura de Software

La arquitectura del software debe modelar la estructura del sistema de cómputo que se va a realizar y la manera en que los datos y los componentes en este sistema interactúan entre sí. La arquitectura permite al ingeniero de software analizar la efectividad del diseño para cumplir con los requisitos establecidos y reducir los riesgos asociados a la construcción del software (14).

3.2.1 Patrón Modelo Vista Controlador

Modelo Vista Controlador es un patrón arquitectónico que separa la lógica de negocio, la interfaz de usuario, y la lógica de control en tres componentes distintos (19). Symfony2 basa su funcionamiento interno en este patrón de arquitectura de software. Cuando el cliente solicita una página o recurso del sitio, el sistema se comporta de la siguiente forma:

1. El sistema de enrutamiento determina qué Controlador está asociado con la página.
2. Symfony2 ejecuta el Controlador asociado.
3. El Controlador solicita al Modelo los datos. El Modelo son las clases PHP especializadas en obtener información, normalmente de una tabla de la base de datos.
4. Con los datos devueltos por el Modelo, el Controlador solicita a la Vista que cree una página mediante una plantilla y que se inserte los datos del Modelo.
5. El Controlador entrega al servidor la página creada por la Vista (29).

3.3 Patrones de diseño GRASP

Los Patrones Generales de Software de Asignación de Responsabilidades GRASP (General Responsibility Assignment Software Patterns) describen principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones (19). En el diseño del sistema se destaca el uso de cinco de estos patrones:

Experto: Este patrón se le aplica a las clases que cuentan con la información necesaria para cumplir una responsabilidad específica (19). Es muy común que el cumplimiento de una responsabilidad requiera información distribuida en otras clases. En el SAGRSI se utiliza este patrón en la forma en que las clases controladoras y entidades trabajan para brindar información específica de un elemento del sistema.

Creador: Este patrón se aplica cuando una clase debe agregar, contener o instanciar objetos de otras clases (19). Todas las clases controladoras cumplen con este patrón. Se aplica este patrón en las clases ArboldeActivos y Nodo. ArboldeActivos contiene un arreglo de nodos los cuales son instanciados para construir el árbol. A su vez la clase Nodo contiene un objeto de Activo, una lista de objetos de las clases Riesgo, Valoración y Dimensión respectivamente.

Controlador: Se le aplica a las clases que tienen la responsabilidad de atender eventos del sistema. Estas clases son utilizadas para atender los eventos del sistema en un mismo escenario de un caso de uso (19). Se pone en práctica en todas las clases controladoras.

Alta cohesión: La cohesión es una medida del grado de focalización de las responsabilidades de un elemento. La alta cohesión se evidencia cuando una clase tiene una responsabilidad moderada en un área funcional (19). Este patrón se emplea en las clases controladoras y entidades.

Bajo acoplamiento: El acoplamiento es la medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo acoplamiento no depende de muchas otras. El Bajo Acoplamiento permite diseñar clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad (19).

3.4 Modelo de Datos.

Teniendo en cuenta los elementos definidos por la MAGERIT para el Análisis y Gestión de Riesgos de Seguridad Informática se diseñó el siguiente modelo de datos, que constituyó la base sobre la cual se implementó el sistema. Este modelo se generó en la base de datos para luego ser importado hacia Symfony2 en forma de relaciones entre clases entidades utilizando el ORM Doctrine.

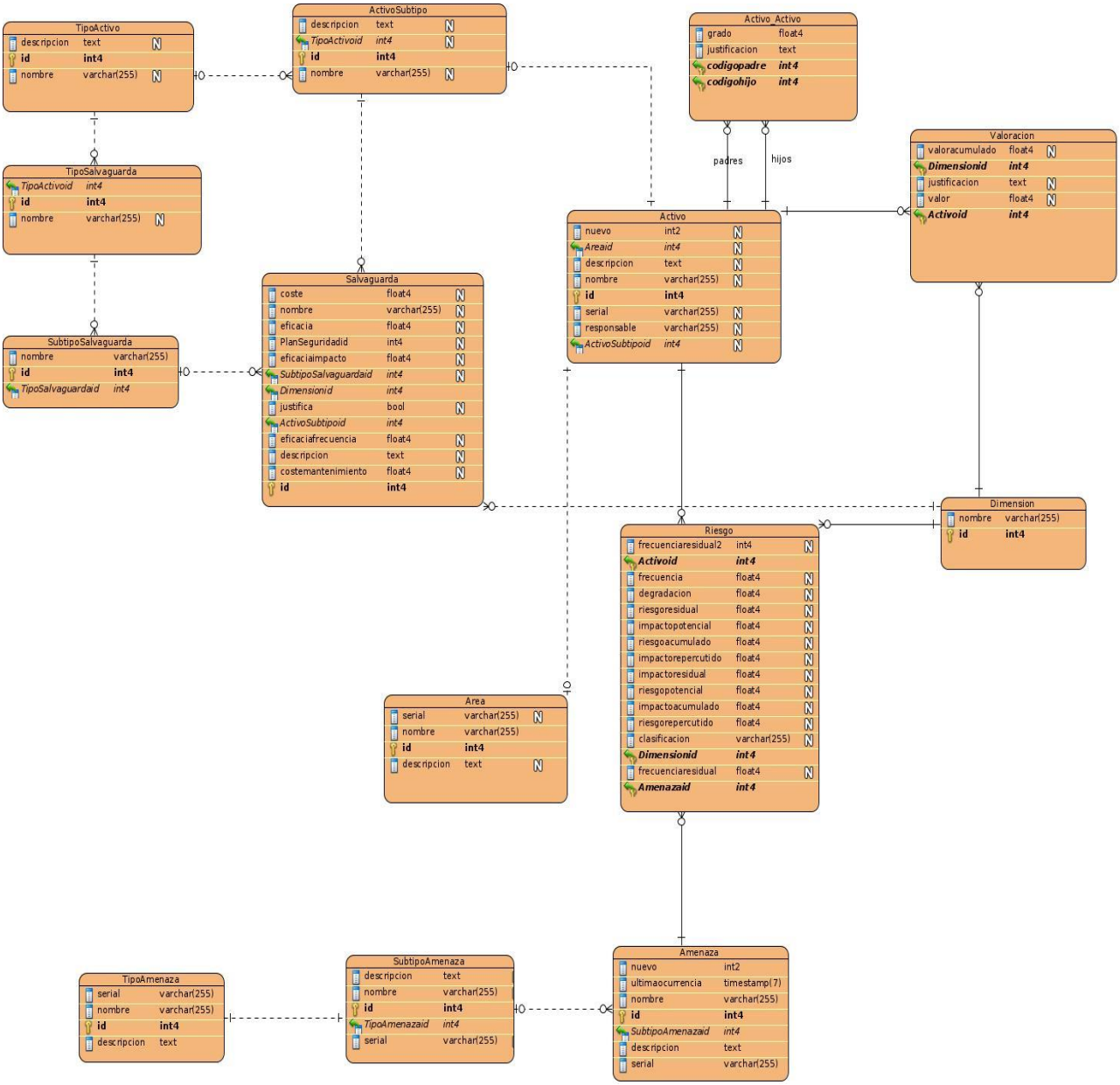

Figura 5: Diagrama Entidad-Relación.

3.5 Tareas de ingeniería

Para implementar las historias de usuario identificadas por el cliente, se efectúa un desglose que deriva en las diferentes tareas de ingeniería a ejecutar por los programadores en la implementación del software. Las tareas de ingeniería garantizan la organización del trabajo debido a que se delegan los responsables de cada tarea. Además, el propio responsable analiza la tarea y re-estima el tiempo de realización de las historias de usuario. El cliente es quien decide en función de sus necesidades las historias a implementar, dejando para iteraciones posteriores aquellas que sobrepasen la capacidad productiva de la iteración (35). A continuación se muestran algunas de las tareas de ingeniería realizadas (Ver Anexo 3).

Tabla 56: Tarea de ingeniería Importar el modelo de la base de datos de SAGRSI y GLPI.

No. de la tarea: 1	**No. de la HU:** 1
Nombre de la tarea: Importar el modelo de la base de datos de SAGRSI y GLPI.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.2
Programador responsable: Roides Cruz Lara	
Descripción: Ejecutar los comandos de Symfony2: doctrine:mapping:import , doctrine:generate:entities.	

Tabla 57: Tarea de ingeniería Identificar los activos del inventario de GLPI

No. de la tarea: 2	**No. de la HU:** 1
Nombre de la tarea: Identificar los activos del inventario de GLPI	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.8
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita cargar los activos reportados en	

GLPI y los elementos asociados a un activo como son el área su categoría y subcategoría de forma lógica.

Tabla 58: Establecer dependencias entre los activos del inventario de GLPI.

No. de la tarea: 3	**No. de la HU:** 2
Nombre de la tarea: Establecer dependencias entre los activos del inventario de GLPI	
Tipo de tarea: desarrollo	**Puntos estimados:** 1
Programador responsable: Roides Cruz Lara	
Descripción: Realizar la funcionalidad que permita establecer las relaciones entre los diferentes activos de GLPI que deben quedar guardadas en la tabla activo_activo.	

Tabla 59: Tarea de ingeniería Gestionar activo.

No. de la tarea: 4	**No. de la HU:** 3
Nombre de la tarea: Gestionar activo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Se implementan las anotaciones de Assert en la entidad Activo. Ejecutar el comando de doctrine:generate:crud.	

3.6 Pruebas

Una vez que se implementa el código del sistema es necesario probarlo para encontrar y enmendar la mayor cantidad de errores antes de entregarlo al cliente. El objetivo de este proceso es diseñar casos de pruebas que tengan una alta probabilidad de encontrar errores. Para alcanzar este objetivo existen técnicas de pruebas de software, que contienen directrices sistemáticas para comprobar la lógica interna y de interfaces de los componentes del sistema, así como los dominios de entrada y salida para identificar errores funcionales y de desempeño. Existen dos formas reconocidas de probar un producto de software: las técnicas de caja negra y las técnicas de caja blanca. Las técnicas de caja negra son las que se aplican en la interfaz del software, examinado aspectos funcionales. Las técnicas de caja blanca se basan en el examen interno y de funcionamiento de los procedimientos del sistema, esta técnica realiza casos de pruebas que ejercitan conjuntos específicos de condiciones, bucles y la combinación de ambos (36).

3.6.1 Pruebas unitarias

Un problema común en los proyectos de software tradicionales es que a la hora de realizar pruebas a nivel de sistema, éste no se ha probado a nivel unitario. Esta manera de detectar errores es extremadamente costosa e insuficiente y puede consumir incluso parte del tiempo planificado para las pruebas de aceptación. XP evita este escenario mediante la automatización de los test unitarios y la continua integración. Los programadores detectan y corrigen los errores unitarios durante las mismas secciones de programación, garantizando que cuando el código llegue las pruebas de aceptación el sistema realice lo que el programador quería, así la prueba de aceptación se puede enfocar en determinar en cuanto el software cumple con las expectativas del cliente (37).Las pruebas unitarias se realizan a una parte del código de la aplicación para comprobar si funciona tal y como debería hacerlo. Normalmente los fragmentos de código son la parte más pequeña posible que se pueda probar, mas en la práctica, se suelen probar clases enteras, a menos que sean muy complejas y haya que probar sus métodos por separado. En Symfony2 cada test se define en una clase cuyo nombre acaba en Test. Se ejecutaron un total de 93 test encontrándose 22 errores, todos resueltos antes de cada entrega del sistema. A continuación se muestra un fragmento del código del test unitario realizado sobre la clase ActivoController (29).

```php
<?php

namespace SAGRSI\AnalisisBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ActivoControllerTest extends WebTestCase
{

    public function testCompleteScenario()
    {
        // Creando un nuevo cliente de la aplicación
        $client = static::createClient();

        // Crear un nuevo activo
        $crawler = $client->request('GET', '/activo/Hardware');
        $this->assertEquals(200, $client->getResponse()->getStatusCode(),
                "Inesperado estado de código HTTP para GET /activo/{tipo}");
        $crawler = $client->click($crawler->selectLink('Añadir activo')->link());

        // Llenando el formaulario y enviando
        $form = $crawler->selectButton('Añadir')->form(array(
            'sagrsi_analisisbundle_activotype[nombre]'  =>'f4-lab208-pc8' ,
            'sagrsi_analisisbundle_activotype[descripcion]'  =>'Pc de laboratorio 202' ,
            'sagrsi_analisisbundle_activotype[tipoactivoid]'  =>'Informárica personal' ,
            'sagrsi_analisisbundle_activotype[area]'  =>'Laboratorios' ,

            // ... other fields to fill
        ));

        $client->submit($form);
        $crawler = $client->followRedirect();
```

Figura 6: Diagrama Test Unitario de la clase ActivoController.

3.6.2 Pruebas de aceptación

Las pruebas de aceptación son aquellas que detectan errores visibles al cliente. Al definir los detalles de las pruebas de aceptación se define la calidad que el cliente espera del sistema. Esto se logra mediante la colaboración entre el cliente y el equipo de desarrollo utilizando como guía las historias de usuario (37). Para la realización de este proceso se diseñaron un total de 21 casos de prueba, que fueron distribuidos y aplicados al final de cada iteración a la funcionalidad correspondiente. Se definió la siguiente plantilla para los casos de prueba de aceptación:

Código: Número de prueba.

Historia de usuario: Número de la historia de usuario a la que se le está realizando el caso de prueba.
Nombre: Descripción más detallada del nombre de la historia de usuario.

Iteración: Iteración a la que pertenece la historia de usuario.
Descripción: Breve descripción del propósito de la prueba.

Condiciones de ejecución: Condiciones especiales que deben tenerse en cuenta para ejecutar el caso de prueba.
Entradas/pasos de ejecución: Entradas o funciones que deben ejecutarse para realizar el caso de prueba.
Resultados esperados: Salida u objetivo que debe cumplir la funcionalidad a la que se le aplica el caso de prueba.
Evaluación de las pruebas: Evaluación de éxito del caso de prueba. Prueba satisfactoria en caso de éxito o prueba insatisfactoria en caso de fallo.

En las siguientes tablas se muestran dos ejemplos de casos de pruebas de aceptación realizadas al sistema, el resto pueden verse en Anexo 4:

Caso de prueba de aceptación Identificar los activos del inventario de GLPI	
Código: 1	**HU:** 1
Nombre: Identificar los activos del inventario de GLPI	
Descripción: Prueba para la funcionalidad que permite identificar los activos reportados en GLPI.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y hacer clic en el botón Identificar activos de GLPI.	
Resultado esperado: El sistema muestra una tabla con los activos reportados en GLPI, su serial, su clasificación según la MAGERIT	
Evaluación de la prueba: Satisfactoria	

Caso de prueba de aceptación Generar informe Modelo del valor.	
Código: 8	**HU:** 8
Nombre: Generar informe Modelo del valor.	

Descripción: Prueba para la funcionalidad Generar el informe Modelo del valor.
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.
Entradas/Pasos de ejecución: Hacer clic en el botón Modelo del Valor. Hacer clic en botón generar informe.
Resultado esperado: El sistema debe mostrar para cada activo su valor propio y acumulado, sus superiores e hijos. Debe exportar a PDF el informe.
Evaluación de la prueba: Satisfactoria

3.6.3 Resultados de las pruebas

Una vez realizadas las pruebas sobre el sistema se detectaron un total de 26 no conformidades clasificadas en significativas, no significativas y recomendaciones. Las no significativas fueron en general errores ortográficos y números que aparecían fuera de lugar en las vistas debido al uso de función *echo* para ver el resultado del código y otras malas prácticas de programación. Entre las más significativas se destacan las siguientes:

1. En la funcionalidad añadir una dependencia directa, cuando se agrega una dependencia la tabla donde se buscan los activos pierde la paginación.
2. En la funcionalidad añadir una dependencia directa cuando se agrega una dependencia no se puede buscar en la tabla.
3. En la funcionalidad crear un riesgo cuando se agrega un activo no se puede buscar en la tabla.
4. En la funcionalidad crear riesgo cuando se agrega una tabla donde se buscan los activos pierde la paginación.
5. En la funcionalidad de evaluar riesgos se repiten los riesgos.
6. Al generar el informe Evaluación de las Salvaguardas el PDF muestra el scroll.

En la siguiente gráfica se muestran las no conformidades encontradas a lo largo de todo el proceso de desarrollo del sistema.

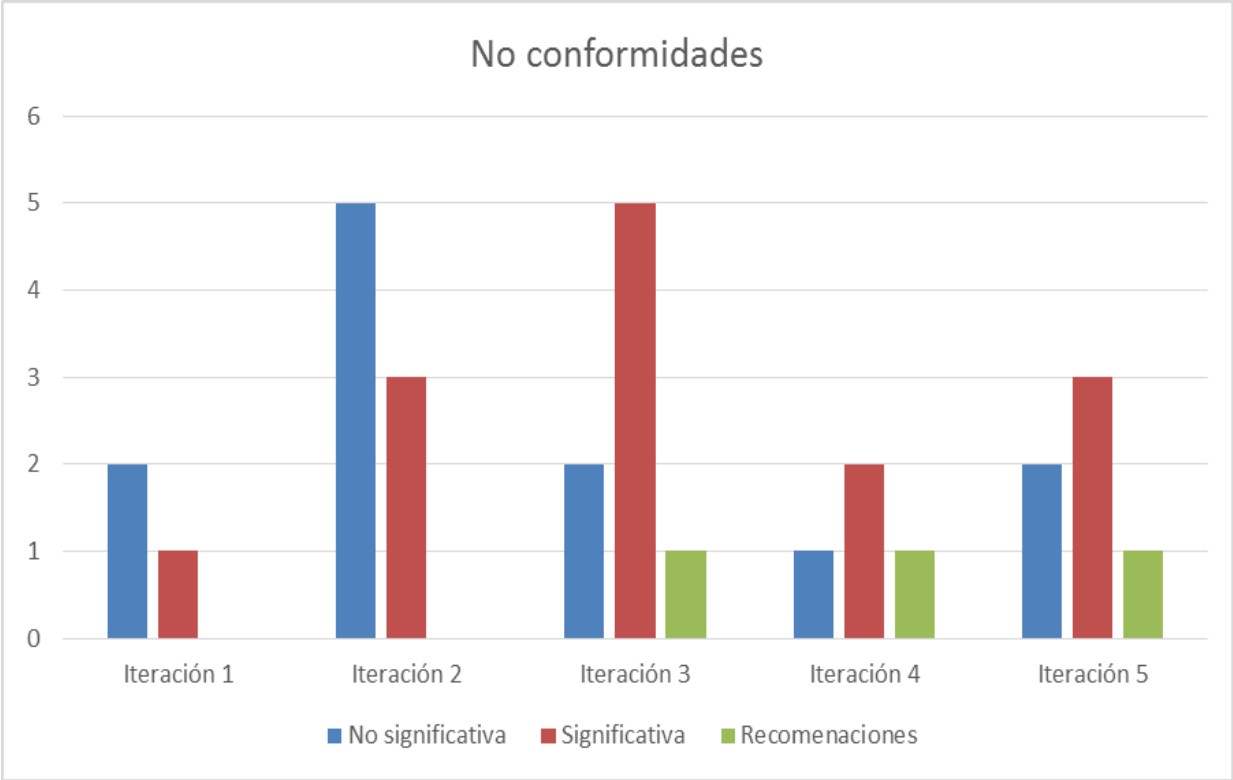

Figura 5: No conformidades por iteraciones.

3.7 Evaluación de beneficios y reducción de complejidad

A continuación se muestra una evaluación de los beneficios obtenidos en el Departamento de Tecnología debido a la utilización del SAGRSI. Para realizar esta evaluación se tuvieron en cuenta los siguientes aspectos:

1. **Importancia estratégica:** corresponde al nivel de relevancia de la aplicación que se está evaluando, mientras más cerca del núcleo del negocio, más crítica será su operación (38).
2. **Cobertura funcional:** A mayor cobertura de los requerimientos funcionales y del negocio, se transforma en una solución más adecuada (38).

3. **Evolución:** Evalúa la capacidad de evolución y el nivel de flexibilidad que presenta el sistema ante cambios futuros en la forma de ejecución del proceso. Estos aspectos van de la mano de la arquitectura y diseño del sistema (38).
4. **Costo:** Evaluar el costo total de propiedad (38).
5. **Tiempos:** Tiempos efectuados en las actividades con la utilización del sistema anterior y el sistema en cuestión (39).
6. **Uso de Estándares:** Uso de estándares y metodologías utilizados en el proceso de Gestión de Riesgos
7. **Reducción de pérdidas de información:** Se refiere a la medida en que se mantiene la integridad de los datos.
8. **Rapidez de adaptación a los cambios:** Velocidad de adaptación ante los cambios en el sistema de información.

Tabla 113: Evaluación de beneficios.

Criterio	Evaluación	Justificación
Importancia estratégica	**7**	Se aplica al proceso de Gestión de Riesgos que tiene una importancia alta dentro de la gestión del departamento.
Cobertura funcional	**8**	El proceso de Gestión de Riesgos comprende 22 tareas divididas en dos subprocesos. SAGRSI implementa 15 tareas, ofreciendo una cobertura funcional alta teniendo en cuenta de que existen tareas que no son automatizables.
Evolución	**5**	En el SAGRSI pueden añadirse nuevas subcategorías de activos, amenazas y salvaguardas, nuevas dimensiones de seguridad informática y nuevas áreas. Solo se puede realizar el Análisis de Riegos por

		el método cuantitativo.
Costo	**10**	Libre de costo.
Tiempos	**9**	El proceso de análisis de riesgos como se realizaba anteriormente tomaba una sesión de trabajo. Se efectuaron mediciones de tiempo utilizando el SAGRSI en las que el proceso quedaba terminado en menos de 50 minutos lo que reduce al 25 por ciento el tiempo de realización del proceso.
Uso de Estándares	**8**	Utiliza la metodología MAGERIT.
Reducción de pérdidas de información	**8**	La información de los activos y amenazas se obtiene de forma automática, la pérdida de información es casi nula en estas tareas. Existen datos que aún se introducen al sistema manualmente como las salvaguardas.
Rapidez de adaptación a los cambios	**9**	Un cambio en una computadora será reportado por OCS a GLPI. Para identificar los cambios solo se debe volver a identificar los activos y SAGRSI actualizará sus datos con los de GLPI.
Evaluación final	**8**	La aplicación cumple con las necesidades del cliente.

Puntuación Máxima: 10 **Puntuación Mínima: 1**

Para conocer en qué medida se logra reducir la complejidad del proceso de Gestión de Riesgos es necesario remitirse en la evaluación anterior a los indicadores de Tiempo y Rapidez de

adaptación a los cambios. Estos dos aspectos fueron seleccionados al inicio de la investigación entre las diferentes fuentes de complejidad que presentaba el proceso, por ser los que más complejidad aportaban. Además estos aspectos son los más propensos a mejora ya que otras fuentes de complejidad como la cantidad de variables involucradas (activos) y sus interacciones, los diferentes estados que se pueden producir, la dinámica de las relaciones entre las variables y la cantidad de información disponible son elementos inherentes al sistema, cuyo tratamiento esta fuera del alcance de la presente investigación.

El tiempo en que se realizaba el proceso fue reducido considerablemente (25 por ciento), esto se debe en gran medida a la automatización de las tareas de Identificar amenazas e Identificar activos, siendo esta última la que más tiempo tomaba realizar. También, tareas como el establecimiento de dependencias indirectas entre activos, la posibilidad que brinda el sistema de valorar varios activos que cumplan con los mismos criterios de valoración, la generación de los informes e incluso la generación de gráficas contribuyeron directamente a reducir el tiempo de realización del proceso. La velocidad de adaptación a los cambios también tiene un aumento significativo como consecuencia directa de la automatización de las tareas de identificación, el SAGRSI identifica los cambios en el sistema tan rápido como estos cambios sean reportados en GLPI, un sistema validado, con tiempos de respuesta altos. Por lo antes planteado se evidencia que mediante la utilización del SAGRSI se reduce la complejidad del Proceso de Gestión de Riesgos en la Facultad 4.

3.9 Conclusiones del capítulo

El Framework Synfony2 permitió liberar al programador de la implementación de la arquitectura Modelo-Vista-Controlador, contribuyendo a reducir el tiempo de desarrollo, así como a eliminar errores de implementación que frecuentemente se comenten a la hora de aplicar este patrón arquitectónico. Con la aplicación de los patrones GRASP se logró resolver problemas comunes que surgen al utilizar el paradigma de la programación orientada a objetos.

El proyecto se llevó a cabo según la planificación establecida en 5 iteraciones, que se debe fundamentalmente por las buenas prácticas precisadas por XP y que fueron cumplidas cabalmente por el equipo de desarrollo y el cliente, realizándose reuniones frecuentes y manteniendo una estrecha comunicación.

Las pruebas fueron realizadas paralelamente al desarrollo del sistema para garantizar la calidad y el cumplimiento de la planificación. Las pruebas unitarias desarrolladas por el

programador eliminaron errores en la lógica interna del sistema, permitiendo que las pruebas de aceptación establecieran el nivel de conformidad o satisfacción del cliente respecto al producto desarrollado.

CONCLUSIONES

Finalizado el presente trabajo y teniendo en cuenta los resultados obtenidos tanto desde el punto de vista investigativo como desde el punto de vista práctico se consideran las siguientes conclusiones:

La propuesta de solución puede entenderse como una implementación de la MAGERIT aunque contiene, desde el punto de vista práctico, novedades a tener en cuenta para realizar la Gestión de Riesgos, sobre todo si se trata de entornos tecnológicos amplios y en constante cambio. EL SAGRSI brinda facilidades que agilizan el proceso de Gestión de Riesgos permitiendo realizar valoraciones a varios activos simultáneamente, siempre y cuando estos cumplan con los mismos criterios de valoración, así el especialista solo debe concretarse en valorar individualmente aquellos activos más importantes para la organización.

La rapidez con la que SAGRSI realiza el inventario de activos, gracias a su integración con las herramientas GLPI y OCS, es un ejemplo de los beneficios que brinda la automatización e integración de los procesos de Gestión de Seguridad Informática.

La identificación de las amenazas, obtenidas de los incidentes reportados a GLPI, ya sea por los propios técnicos o por los clientes OCS propician un conocimiento de los eventos negativos, que verdaderamente han ocurrido en nuestro sistema de información, visto desde el enfoque de la Seguridad Informática. Luego la estimación de la probabilidad de materialización de las amenazas, está basada en datos reales del sistema de información, por tanto se realiza un análisis más preciso que permite enfocar los esfuerzos de protección en función de las amenazas comunes al sistema en cuestión.

RECOMENDACIONES

Durante el transcurso de la investigación fueron surgiendo diferentes interrogantes e ideas que pueden ser, a consideración de los autores, la base de futuras investigaciones y trabajos:

La tarea de selección de las salvaguardas es un proceso de toma de decisiones en el que se deben tomar en cuenta diferentes aspectos como el costo, el beneficio, su eficacia y otra serie de elementos sobre su operación y monitorización. Se recomienda investigar como un sistema informático podría servir de herramienta para guiar este proceso, recomendando las mejores salvaguardas a aplicar, teniendo en cuenta los aspectos planteados anteriormente y la utilización de bases de conocimiento.

La Gestión de Riesgos es un proceso cíclico y continuo, es por eso que sería de utilidad implementar un Historial de los Proyectos de Análisis y Gestión de Riesgos para así tener una referencia de que tan precisa fue estimación realizada y garantizar la mejora continua del proceso.

Se recomienda implementar funcionalidades en GLPI y SAGRSI que admitan una comunicación bidireccional entre estos sistemas. Las valoraciones sobre los activos realizadas en SAGRSI, podrían servir a los técnicos para conocer qué tan importante es el activo para la Seguridad Informática, sirviendo de indicador para establecer prioridades en el servicio de soporte técnico.

En la medida que la investigación se fue desarrollando se ha podido apreciar la importancia de la integración entre las diferentes aplicaciones informáticas, ante todo porque aumenta la sinergia del Sistema de Gestión de Seguridad Informática, es por eso que se recomienda continuar la integración del SAGRSI con otros sistemas ya implementados o en desarrollo como son: el Sistema para el Control de Acceso a los Laboratorios del Centro de Tecnologías para la Formación (SCAFORTES), el Owncloud y Ossec. La integración podría llevarse a un nivel superior mediante el desarrollo de una suite de aplicaciones que abarque gran parte de los procesos automatizables de Seguridad Informática.

REFERENCIAS

1. **ITGI.** *Control Objectives for Information and Related Technology (COBIT 4.1).* s.l. : IT Governance Institute (ITGI), 2007.

2. **ISO/IEC.** *ISO/IEC 27000: Information technology — Security techniques— Information security management systems — Overview and Vocabulary.* s.l. : International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2014.

3. **Perurena, Raydel Montesino.** *Modelo para la Gestión Automarizada e Integrada de Controles de Seguridad Informática.* s.l. : Universidad de las Ciencias Informáticas (UCI), 2012.

4. **Bar-Yam, Yaneer.** *Dynamics of Complex Systems.* Massachusetts : Addison-Wesley, 1992.

5. **ALVAREZ, ALFONSO CORNEJO.** *COMPLEJIDAD Y CAOS: GUÍA PARA LA ADMINISTRACIÓN DEL SIGLO XXI* . 2004.

6. **CSAE: Consejo Superior de Administración Electrónica.** *MAGERIT – versión 2: Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información.* s.l. : Ministerio de Administraciones Públicas, 2006.

7. **ISO/IEC.** *ISO/IEC 27001: Information technology - Security techniques - Information security management systems - Requirements.* s.l. : International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2014.

8. **Amutio Gómez, Miguel Angel, Candau, Javier and Antonio Mañas, José .** *MAGERIT – versión 3.0. Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información. Libro I - Método.* Madrid : Ministerio de Hacienda y Administraciones Públicas Secretaría General Técnica Subdirección General de Información, Documentación y Publicaciones Centro de Publicaciones, 2012.

9. ISO27000.es. *El portal de ISO 27001 en español. Gestión de Seguridad de la Información.* [Online] enero 2, 2015. http://iso27000.es/sgsi.html.

10. iso-27005. *www.27000.org.* [Online] enero 2, 2015. http://www.27000.org/iso-27005.htm.

11. R-box. [Online] [Cited: febrero 5, 2015.] http://www.r-box.com.ar/que_es_r-box/.

12. EAR / PILAR. [Online] [Cited: enero 20, 2015.] http://www.ar-tools.com/es/index.html.

13. **Avison, D.E. y Fitzgerald, G.** *Information Systems Development: Methodologies, Techniques, and Tools.* s.l. : McGraw-Hill, 1995.

14. **Pressman, Roger.** *Ingeniería del Software, Un Enfoque Práctico.* México : McGraw-Hill/INTERAMERICANA DE MEXICO, 2005. SBN: 9789701054734.

15. **Beck, Kent.** *Extreme Programming Explained.* s.l. : Addison Wesley, 1999.

16. What is Scrum. *www.scrum.org.* [Online] [Cited: enero 4, 2015.] https://www.scrum.org/Resources/What-is-Scrum.

17. **Jacobson, Ivar, Boch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.

18. Ediciones. *http://www.visual-paradigm.com.* [Online] [Cited: febrero 4, 2015.] http://www.visual-paradigm.com/editions/standard.jsp.

19. **Larman, Craig.** *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado.Segunda Edición.* Madrid : PRENTICE HALL, 2003. ISBN:84-205-3438-2.

20. **Bakken, Stig Sæther, Aulbach, Alexander and Schmid, Egon.** *PHP Manual.* s.l. : The PHP Documentation Group, 2004.

21. **Kabir, Mohamend.** *La Biblia Server Apache 2 .* s.l. : Wiley, 2002.

22. **Musciano, Chuck and Kemedy, Bill.** *HTML la guía completa.* s.l. : MacGraw-Hill, 1999.

23. **Gauchat, Juan Diego.** *El gran libro de HTML5, CSS3 y Javascript .* Barcelona : Marcombo, 2012. ISBN:978-84-267-1782-5 .

24. **Brandenbaugh, Jerry.** *Aplicaiones JavaScript.* s.l. : O'Reilly & Associates, 2000.

25. **Pérez, Javier Eguíluz.** *Introducción a JavaScript.* 2008.

26. **Zeldman, Jeffrey.** *Diseño con estándares Web.* s.l. : ANAYA, 2004.

27. **Pérez, Javier Eguíluz.** *Introducción al CSS.* s.l. : Creative Commons Reconocimiento, 2008.

28. Documentation, Training & Support . *www.netbeans.org.* [Online] [Cited: febrero 5, 2014.] https://netbeans.org/kb/index.html.

29. **Eguiluz, Javier.** *Desarrollo web ágil con Symfony 2 .* 2013.

30. **Otto, Mark and Thornton , Jacob .** Bootstrap 3, el manual oficial. *LibrosWeb.es.* [Online] 2007. [Cited: 1 27, 2015.] https://librosweb.es/libro/bootstrap_3/.

31. Bootstrap Tutorial. *Tutorialspoint.com.* [Online] [Cited: 1 27, 2015.] http://www.tutorialspoint.com/bootstrap/bootstrap_tutorial.pdf.

32. **Gilfillan, Ian.** *La Biblia MySQL.* s.l. : SYBEX, 2006.

33. **Zapata, Manuel.** *Aplicación GLPI (Gestionnaire Libre de Parc Informatiqué) Para un Centro de Atención al Usuario (CAU) Instalación y Configuración .* 2010.

34. **Casas, Sandra y Reinaga, Héctor.** *Identificación y Modelado de Aspectos Tempranos dirigido por Tarjetas de Responsabilidades y Colaboraciones.* Río Gallegos : Universidad Nacional de la Patagonia Austral, 2014.

35. **Ariel Erlijman Piwen, Alejandro Goyén Fros.** *Problemas y Soluciones en la implementacion de Extreme Programming.* 2001.

36. **Presman, Roger.** Capítulo 14 Técnicas de Prueba de Softwares . *Ingeniería del Software, Un Enfoque Práctico 5ta Edición.* s.l. : MacGraw-Hill, 2002.

37. **Lisa Crispin, Tip House.** *Testing Extreme Programming.* s.l. : Addison Wesley, 2002.

38. **Barros, Alejandro.** *Modelo de Análisis Costos-Beneficio para Sistemas Integrados de Administración Financiera.* s.l. : Banco Interamericano de Desarrollo, 2012.

39. **Suhail, Lizbeth Castro Gutierrez Vanina and Gina Maza Anton.** DESARROLLO DEL SISTEMA DE INFORMACIÓN ACADÉMICO DEL INSTITUTO SUPERIOR TECNOLÓGICO MANUEL YARLEQUE ESPINOZA DE CATACAOS, UTILIZANDO LA METODOLOGÍA UML. *http://www.eumed.net.* [Online] [Cited: mayo 20, 2015.] http://www.eumed.net/libros-gratis/2010b/683/Costos%20y%20beneficios%20del%20sistema%20propuesto.htm.

40. **Stoneburner, Gary , Goguen, Alice and Feringa, Alexis.** *NIST Special Publication 800-30: Risk Management Guide for Information Technology Systems.* s.l. : National Institute of Standars and Technology (NIST) , 2002.

41. **Palacios, Juan.** *Flexibilidad con Scrum.* 2007.

42. *METODOLOGIAS DE GESTION DE RIESGOS (OCTAVE, MAGERIT, DAFP).* **OCHOA, BLANCA RUBIELA DUQUE.** s.l. : UNIVERSIDAD DE CALDAS, 2012.

43. **MIC.** *Resolución 127/2007 MIC. Reglamento de seguridad para las tecnologías de la información. .* s.l. : Ministerio de la informática y las comunicaciones (MIC), 2007.

44. **Buschmann, Frank, Henney, Kevlin y Schmidt, Douglas C.** *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages Volumen 5 de Pattern-Oriented Software Architecture.* s.l. : John Wiley & Sons, 2007. ISBN: 0470512571, 9780470512579.

ANEXOS

Anexo 1

Tabla 2: HU- Identificar los activos del inventario de GLPI.

Historia de Usuario	
Nombre: Identificar los activos del inventario de GLPI.	**Número:** 1
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1
Descripción: El sistema carga y muestra los activos de información no esenciales del inventario de GLPI con la clasificación según MAGERIT.	
Observación:	

Tabla 3: HU- Establecer dependencias entre los activos del inventario de GLPI

Historia de Usuario	
Nombre: Establecer dependencias entre los activos del inventario de GLPI	**Número:** 2
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1
Descripción: El sistema muestra los activos de información no esenciales del inventario de GLPI con las dependencias identificadas en el inventario de GLPI	
Observación: El grado de dependencia tendrá valor por defecto 1.	

Tabla 4: HU- Gestionar activo.

Historia de Usuario	
Nombre: Gestionar activos.	**Número:** 3
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al usuario crear, editar, mostrar y eliminar todos los datos de un activo.	
Observación: Los datos que se guardan de cada activo están descritos en el Catálogo de Elementos de la MAGERIT	

Tabla 5: HU- Valorar activo.

Historia de Usuario	
Nombre: Valorar activo.	**Número:** 4
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al usuario valorar los activos en las diferentes dimensiones de seguridad informática.	
Observación: Los parámetros y las dimensiones para valorar un activo están descritos en el Catálogo de Elementos de la MAGERIT.	

Tabla 6: HU- Establecer dependencias directas entre activos.

Historia de Usuario	
Nombre: Establecer dependencias directas entre activos.	**Número:** 5

Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al administrador establecer dependencias directas entre activos de información así como el grado de estas.	
Observación:	

Tabla 7: HU- Establecer dependencias indirectas entre activos.

Historia de Usuario	
Nombre: Establecer dependencias indirectas entre activos.	**Número:** 6
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1
Descripción: El sistema debe establecer dependencias indirectas entre activos de información así como el grado de estas.	
Observación: La forma en que se calcula el grado de dependencias indirectas está descrita en la Guía de Técnicas de MAGERIT.	

Tabla 8: HU- Calcular valor acumulado del activo.

Historia de Usuario	
Nombre: Calcular valor acumulado del activo.	**Número:** 7
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 0.5

Descripción: El sistema debe calcular el valor acumulado sobre un activo.
Observación: La forma en que se calcula el valor acumulado está descrita en la Guía de Técnicas de MAGERIT.

Tabla 9: HU- Generar informe Modelo del valor.

Historia de Usuario	
Nombre: Generar informe Modelo del valor.	**Número:** 8
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1
Descripción: El sistema debe mostrar el árbol de activos especificando para cada activo su valor propio, su valor acumulado y sus dependencias.	
Observación: Los datos mostrados en este informe están descritos en el Catálogo de Elementos de la MAGERIT.	

Tabla 10: HU- Identificar las amenazas desde GLPI.

Historia de Usuario	
Nombre: Identificar las amenazas desde GLPI.	**Número:** 9
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Media	**Punto Estimado:** 0.5
Descripción: El sistema debe mostrar las amenazas que han sido reportadas en las incidencias de GLPI.	
Observación:	

Tabla 11: HU- Gestionar amenazas.

Historia de Usuario	
Nombre: Gestionar amenazas.	**Número:** 10
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al usuario crear, editar, mostrar y eliminar todos los datos de una amenaza.	
Observación: Los datos que se almacenan de una amenaza aparecen descritos en el Libro II-Catálogo de elementos de MAGERIT.	

Tabla 12: HU- Relacionar las amenazas con los activos que afecta.

Historia de Usuario	
Nombre: Relacionar las amenazas con los activos que afecta.	**Número:** 11
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Media	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al administrador relacionar las amenazas con los activos que afecta.	
Observación:	

Tabla 13: HU- Calcular impacto repercutido y acumulado de una amenaza.

Historia de Usuario	
Nombre: Calcular impacto repercutido y acumulado de una amenaza.	**Número:** 12
Usuario: Administrador	
Prioridad: Alta	**Iteración:** 2
Complejidad: Media	**Punto Estimado:** 0.5
Descripción: El sistema debe calcular el impacto repercutido de una amenaza en el árbol de activos	
Observación:	

Tabla 14: HU- Generar informe Mapa de Riesgos.

Historia de Usuario	
Nombre: Generar informe Mapa de Riesgos.	**Número:** 13
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 1
Complejidad: Alta	**Punto Estimado:** 1
Descripción: El sistema debe generar un informe de las amenazas posibles, caracterizadas por su frecuencia de ocurrencia y la degradación que causarían en los activos	
Observación: Ver catálogo de elementos de la MAGERIT	

Tabla 15: HU- Gestionar salvaguarda.

Historia de Usuario	
Nombre: Gestionar salvaguarda.	**Número:** 14

Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al administrador crear, editar, mostrar y eliminar todos los datos de una salvaguarda.	
Observación: Los datos que se almacenan de una salvaguarda aparecen descritos en el Libro II-Catálogo de elementos de MAGERIT.	

Tabla 16: HU- Calcular eficacia de un paquete de salvaguardas.

Historia de Usuario	
Nombre: Calcular eficacia de un paquete de salvaguardas	**Número:** 15
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Baja	**Punto Estimado:** 1
Descripción: El sistema debe calcular la eficacia de un paquete de salvaguardas acumuladas sobre un activo.	
Observación: La forma en que se calcula este valor se encuentra definida en la Guía de técnicas de la MAGERIT.	

Tabla 17: HU- Aplicar una salvaguarda.

Historia de Usuario	
Nombre: Aplicar una salvaguarda.	**Número:** 16
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 2

Complejidad: Media	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir al usuario aplicar una salvaguarda a un tipo de activo.	
Observación:	

Tabla 18: HU- Generar informe Evaluación de salvaguardas.

Historia de Usuario	
Nombre: Generar informe Evaluación de salvaguardas.	**Número:** 17
Usuario: Usuario	
Prioridad: Media	**Iteración:** 2
Complejidad: Alta	**Punto Estimado:** 0.5
Descripción: El sistema debe Generar informe Evaluación de salvaguardas desplegadas y caracterizadas por su grado de efectividad.	
Observación:	

Tabla 19: HU- Estimar riesgo acumulado.

Historia de Usuario	
Nombre: Estimar riesgo acumulado.	**Número:** 18
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 3
Complejidad: Baja	**Punto Estimado:** 0.2
Descripción: El sistema debe calcular el riesgo acumulado.	
Observación:	

Tabla 20: HU- Estimar riesgo repercutido.

Historia de Usuario	
Nombre: Estimar riesgo repercutido.	**Número:** 19
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 3
Complejidad: Baja	**Punto Estimado:** 0.2
Descripción: El sistema debe calcular el riesgo repercutido.	
Observación:	

Tabla 21: HU- Estimar riesgo residual.

Historia de Usuario	
Nombre: Estimar riesgo residual.	**Número:** 20
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 3
Complejidad: Baja	**Punto Estimado:** 0.2
Descripción: El sistema debe calcular el riesgo residual.	
Observación:	

Tabla 22: HU- Estimar riesgo potencial.

Historia de Usuario	
Nombre: Estimar riesgo potencial.	**Número:** 21
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 3

Complejidad: Baja	**Punto Estimado:** 0.2
Descripción: El sistema debe calcular el riesgo potencial.	
Observación:	

Tabla 23: HU- Generar informe Estado del Riesgo

Historia de Usuario	
Nombre: Generar informe Estado del Riesgo	**Número:** 22
Usuario: Usuario	
Prioridad: Alta	**Iteración:** 3
Complejidad: Alta	**Punto Estimado:** 1
Descripción: Informe que detalla para cada activo los riesgos, potenciales, acumulados, repercutidos y residuales, ocasionados por las amenazas.	
Observación:	

Tabla 24: HU- Gestionar riesgos.

Historia de Usuario	
Nombre: Gestionar riesgos.	**Número:** 23
Usuario: Administrador	
Prioridad: Media	**Iteración:** 3
Complejidad: Alta	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir realizar un tratamiento de los riesgos como: eliminar el riesgo eliminando sus causas: información tratada, servicios prestados, arquitectura del sistema, reducir o limitar el impacto, reducir la probabilidad de que la amenaza ocurra.	

Observación:

Tabla 25: HU- Autenticar usuario.

Historia de Usuario	
Nombre: Autenticar usuario.	**Número:** 24
Usuario: Administrador, Usuario	
Prioridad: Alta	**Iteración:** 2
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir la autenticación una vez que el usuario introduzca su usuario y contraseña correctamente.	
Observación:	

Tabla 26: HU- Gestionar usuario.

Historia de Usuario	
Nombre: Gestionar usuario.	**Número:** 25
Usuario: Administrador	
Prioridad: Alta	**Iteración:** 2
Complejidad: Baja	**Punto Estimado:** 0.5
Descripción: El sistema debe permitir crear, mostrar, actualizar, eliminar los datos de un usuario.	
Observación:	

Anexo 2

Tabla 29: Tarjeta CRC ActivoController

Clase: ActivoController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a Activos. Listar activos.	Activo.

Tabla 30: Tarjeta CRC Activo

Clase: Activo	
Responsabilidades	**Colaboraciones**
Modelar el activo de información y su lógica de negocio. Obtener propiedades de un activo. Modificar propiedades de un activo.	ActivoSubtipo Doctrine Mapping ORM

Tabla 31: Tarjeta CRC ActivoSubtipoController

Clase: ActivoSubtipoController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a ActivoSubtipo. Listar subtipos de activo.	ActivoSubtipo.

Tabla 32: Tarjeta CRC ActivoSubtipo

Clase: ActivoSubtipo	
Responsabilidades	**Colaboraciones**
Modelar el subtipo activo de información y su lógica de negocio. Obtener propiedades de un subtipo de activo.	TipoActivo Doctrine Mapping ORM

Modificar propiedades de un subtipo de activo.	

Tabla 33: Tarjeta CRC TipoActivoController

Clase: TipoActivoController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a TipoActivo Listar tipos de activo.	TipoActivo.

Tabla 34: Tarjeta CRC TipoActivo

Clase: TipoActivo	
Responsabilidades	**Colaboraciones**
Modelar el tipo de activo de información. Obtener propiedades de un tipo de activo. Modificar propiedades de un tipo de activo.	TipoActivo Doctrine Mapping ORM

Tabla 35: Tarjeta CRC ActivoActivoController

Clase: ActivoActivoController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a ActivoActivo. Generar informe Modelo del Valor	ActivoActivo

Tabla 36: Tarjeta CRC ActivoActivo

Clase: ActivoActivo	
Responsabilidades	**Colaboraciones**
Modelar la relación entre dos activos de información Obtener propiedades de la relación entre	Activo Doctrine Mapping ORM

dos activos. Modificar propiedades de la relación entre dos activos.	

Tabla 37: Tarjeta CRC Arboldeactivo

Clase: Arboldeactivo	
Responsabilidades	**Colaboraciones**
Modelar al sistema de información como un árbol de dependencias de activos de información. Calcular dependencias indirectas entre activos. Calcular valor acumulado de un activo. Calcular riesgo repercutido.	Nodo

Tabla 38: Tarjeta CRC Nodo

Clase: Nodo	
Responsabilidades	**Colaboraciones**
Modelar al activo como parte orgánica del sistema de información con sus dependencias (activos hijos y superiores), valoraciones, riesgos etc. Obtener todas las propiedades de un nodo del árbol de activos.	Activo ActivoActivo Valoración Riesgo Amenaza Salvaguarda

Tabla 39: Tarjeta CRC AmenazaController

Clase: AmenzaController	
Responsabilidades	**Colaboraciones**

Realizar CRUD a Amenaza Listar amenazas por tipos Generar informe Mapa de Riesgos	Amenaza Arboldeactivo.

Tabla 40: Tarjeta CRC Amenaza

Clase: Amenaza	
Responsabilidades	**Colaboraciones**
Modelar la amenaza y su lógica de negocio. Obtener propiedades de una amenaza. Modificar propiedades una amenaza.	Doctrine Mapping ORM

Tabla 41: Tarjeta CRC SubtipoAmenazaController

Clase: SubtipoAmenzaController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a SubtipoAmenaza Listar subtipos de amenaza.	SubtipoAmenaza

Tabla 42: Tarjeta CRC SubtipoAmenaza

Clase: SubtipoAmenaza	
Responsabilidades	**Colaboraciones**
Obtener propiedades de un subtipo de amenaza. Modificar propiedades de un subtipo de amenaza.	TipoAmenza Doctrine ORM

Tabla 43: Tarjeta CRC TipoAmenazaController

Clase: TipoAmenzaController	
Responsabilidades	**Colaboraciones**

Realizar CRUD a TipoAmenaza Listar tipos amenazas	TipoAmenaza

Tabla 44: Tarjeta CRC TipoAmenaza

Clase: TipoAmenaza	
Responsabilidades	**Colaboraciones**
Obtener propiedades de un subtipo de amenaza. Modificar propiedades de un subtipo de amenaza.	Doctrine ORM

Tabla 45: Tarjeta CRC SalvaguardaController

Clase: SalvaguardaController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a Salvaguarda Listar salvaguardas Generar informe Evaluación de las Salvaguardas	Salvaguarda

Tabla 46: Tarjeta CRC Salvaguarda

Clase: Salvaguarda	
Responsabilidades	**Colaboraciones**
Modelar la salvaguarda y su lógica de negocio. Obtener propiedades de una salvaguarda. Modificar propiedades una salvaguarda.	Doctrine Mapping ORM

Tabla 47: Tarjeta CRC SubtipoSalvaguardaController

Clase: SubtipoSalvaguardaController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a SubtipoSalvaguarda Listar salvaguardas Generar Informe Evaluación de las Salvaguardas.	SubtipoSalvaguarda

Tabla 48: Tarjeta CRC SubtipoSalvaguarda

Clase: SubtipoSalvaguarda	
Responsabilidades	**Colaboraciones**
Modelar el subtipo de salvaguarda y su lógica de negocio. Obtener propiedades de una salvaguarda. Modificar propiedades una salvaguarda.	TipoSalvaguarda Doctrine Mapping ORM

Tabla 49: Tarjeta CRC TipoSalvaguardaController

Clase: TipoSalvaguardaController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a TipoSalvaguarda Listar salvaguardas	TipoSalvaguarda

Tabla 50: Tarjeta CRC TipoSalvaguarda

Clase: Salvaguarda	
Responsabilidades	**Colaboraciones**
Modelar el tipo de salvaguarda y su lógica de negocio.	Doctrine Mapping ORM

Obtener propiedades de una salvaguarda. Modificar propiedades una salvaguarda.	

Tabla 51: Tarjeta CRC RiesgoController

Clase: RiesgoController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a Riesgo Listar riesgos Generar Informe Estado del Riesgo	Riesgo Arboldeactivo

Tabla 52: Tarjeta CRC Riesgo

Clase: Riesgo	
Responsabilidades	**Colaboraciones**
Modelar el resigo y su lógica de negocio. Obtener propiedades de un riesgo Modificar propiedades un riesgo	Doctrine Mapping ORM

Tabla 53: Tarjeta CRC ValoraciónController

Clase: ValoraciónController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a Valoración Listar valoraciones	Valoración

Tabla 54: Tarjeta CRC Valoración

Clase: Valoración	
Responsabilidades	**Colaboraciones**

Modelar la valoración y su lógica de negocio. Obtener propiedades de una valoración Modificar propiedades una valoración.	Activo Dimensión Doctrine Mapping ORM

Tabla 55: Tarjeta CRC DimensiónController

Clase: DimensiónController	
Responsabilidades	**Colaboraciones**
Realizar CRUD a Dimensión Listar dimensiones	Dimensión

Tabla 56: Tarjeta CRC Dimensión

Clase: Dimensión	
Responsabilidades	**Colaboraciones**
Modelar la dimensión de seguridad y su lógica de negocio. Obtener propiedades de una dimensión Modificar propiedades una dimensión.	Doctrine Mapping ORM

Tabla 55: Tarjeta CRC DefaultController

Clase: DefaultController	
Responsabilidades	**Colaboraciones**
Proporcionar información general del sistema. Ofrecer la cantidad total de activos, amenazas, salvaguardas y riesgos. Cargar catálogo de elementos del sistema. Debe permitir la configuración del	Activo Amenaza Salvaguarda Riesgo ActivoSubtipo TipoActivo SubtipoAmenaza

sistema.	TipoAmenaza SubtipoSalvaguarda TipoSalvaguarda

Anexo 3

Tabla 56: Tarea de ingeniería Importar el modelo de la base de datos de SAGRSI y GLPI.

No. de la tarea: 1	**No. de la HU:** 1
Nombre de la tarea: Importar el modelo de la base de datos de SAGRSI y GLPI	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.2
Programador responsable: Roides Cruz Lara	
Descripción: Ejecutar los comandos de doctrine:mapping:import, doctrine:generate:entities.	

Tabla 57: Tarea de ingeniería Identificar los activos del inventario de GLPI

No. de la tarea: 2	**No. de la HU:** 1
Nombre de la tarea: Identificar los activos del inventario de GLPI	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.8
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita cargar los activos reportados en GLPI y los elementos asociados a un activo como son el área su categoría y subcategoría de forma lógica.	

Tabla 58: Tarea de ingeniería Establecer dependencias entre los activos del inventario de GLPI

No. de la tarea: 3	**No. de la HU:** 2
Nombre de la tarea: Establecer dependencias entre los activos del inventario de GLPI	
Tipo de tarea: desarrollo	**Puntos estimados:** 1
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita establecer las relaciones entre los diferentes activos de GLPI que deben quedar guardadas en la tabla activo_activo.	

Tabla 59: Tarea de ingeniería Gestionar activo.

No. de la tarea: 4	**No. de la HU:** 3
Nombre de la tarea: Gestionar activo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Se implementan las anotaciones de Assert en la entidad Activo. Ejecutar el comando de doctrine:generate:crud.	

Tabla 60: Tarea de ingeniería Aplicar estilos a las vistas del CRUD de Activo

No. de la tarea: 5	**No. de la HU:** 3
Nombre de la tarea: Aplicar estilos a las vistas del CRUD de Activo	

Tipo de tarea: desarrollo	**Puntos estimados:** 0.8
Programador responsable: Roides Cruz Lara	
Descripción: Aplicar los estilos y JavaScript utilizando el framework Bootstrap	

Tabla 61: Tarea de ingeniería Valorar activos.

No. de la tarea: 6	**No. de la HU:** 4
Nombre de la tarea: Valorar activos	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita realizar valoraciones a varios activos que cumplan con los mismos valores en los criterios de valoración.	

Tabla 62: Tarea de ingeniería Establecer dependencias directas entre activos

No. de la tarea: 7	**No. de la HU:** 5
Nombre de la tarea: Establecer dependencias directas entre activos	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita desde la vista de Editar activo agregar mostrar, editar y eliminar las dependencias directas entre activos mediante la generación del CRUD de la entidad ActivoActivo.	

Tabla 63: Tarea de ingeniería Implementar la clase Nodo

No. de la tarea: 8	**No. de la HU:** 6
Nombre de la tarea: Implementar la clase Nodo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Un nodo contiene un activo, una lista de riesgos, una lista de superiores, una lista de hijos, una lista de valoraciones y los métodos correspondientes para acceder y modificar estos atributos.	

Tabla 64: Tarea de ingeniería Implementar la clase Árbol de Activos

No. de la tarea: 9	**No. de la HU:** 6
Nombre de la tarea: Implementar la clase Árbol de Activos.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Un árbol de activos contiene una lista de Nodos, se instancia con todos los activos, riesgos, valoraciones, dependencias establecidas en la base de datos.	

Tabla 65: Tarea de ingeniería Establecer dependencias indirectas entre activos

No. de la tarea: 10	**No. de la HU:** 6

Nombre de la tarea: Establecer dependencias indirectas entre activos.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Implementar un método capaz de recorrer el árbol de activos estableciendo las dependencias indirectas y el grado de éstas mediante la fórmula definida en la Guía de Técnicas de la MAGERIT. Es un método de la clase Árbol de Activos.	

Tabla 66: Tarea de ingeniería Calcular valor acumulado del activo

No. de la tarea: 11	**No. de la HU:** 7
Nombre de la tarea: Calcular valor acumulado del activo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Implementar el método que calcule el valor que se acumula en los activos desde sus activos superiores por la forma establecida en la Guía de Técnicas de la MAGERIT.	

Tabla 67: Tarea de ingeniería Implementar el controlador del Modelo del Valor

No. de la tarea: 12	**No. de la HU:** 8
Nombre de la tarea: Implementar el controlador del Modelo del Valor	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	

Descripción: El controlador carga todos los activos, dependencias, valoraciones y crea un árbol de activos luego llama al método de calcula el valor acumulado. Se guardan las los cambios realizados por el recorrido en la base de datos y devuelve una vista con una lista de nodos.

Tabla 68: Tarea de ingeniería Implementar la vista del Modelo del Valor

No. de la tarea: 13	**No. de la HU:** 8
Nombre de la tarea: Implementar la vista del Modelo del Valor	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Es una tabla con todos los activos, su valor propio y acumulado en cada dimensión de seguridad. Debe permitir búsquedas generales. Debe permitir exportar a PDF.	

Tabla 69: Tarea de ingeniería Identificar las amenazas de GLPI

No. de la tarea: 14	**No. de la HU:** 9
Nombre de la tarea: Identificar las amenazas de GLPI	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita identificar las amenazas en GLPI, creando riesgos en el activo que impacta y se calcula la frecuencia de ocurrencia frecuencia. Se debe actualizar GLPI con el catálogo de elementos que usa SAGRSI.	

Tabla 70: Tarea de ingeniería Gestionar Amenaza

No. de la tarea: 15	**No. de la HU:** 10
Nombre de la tarea: Gestionar Amenaza	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se implementan las anotaciones de Assert en la entidad Amenaza. Se ejecuta el comando doctrine:generate:crud a la entidad Amenaza	

Tabla 71: Tarea de ingeniería Aplicar estilos al CRUD de Amenaza

No. de la tarea: 16	**No. de la HU:** 10
Nombre de la tarea: Aplicar estilos al CRUD de Amenaza	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Aplicar estilos y funcionalidades JavaScript en las vistas del CRUD de Amenaza utilizando el framework Bootstrap.	

Tabla 72: Tarea de ingeniería Relacionar activos con amenazas

No. de la tarea: 17	**No. de la HU:** 11

Nombre de la tarea: Relacionar activos con amenazas	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Implementar la vista y el controlador que permitan relacionar una amenaza con los activos que ésta afecta, generando un riesgo potencial. Debe permitir realizar búsquedas generales sobre el inventario de activos.	

Tabla 73: Tarea de ingeniería Calcular impacto repercutido de una amenaza

No. de la tarea: 18	**No. de la HU:** 12
Nombre de la tarea: Calcular impacto repercutido de una amenaza	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Implementar el método que recorre el árbol desde el nodo afectado por sus superiores actualizando los riesgos con el impacto repercutido.	

Tabla 74: Tarea de ingeniería Calcular impacto acumulado de una amenaza

No. de la tarea: 19	**No. de la HU:** 12
Nombre de la tarea: Calcular impacto acumulado de una amenaza.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Implementar el método que recorre el árbol desde el nodo hasta sus	

inferiores acumulado impacto sobre estos.

Tabla 75: Tarea de ingeniería Implementar el controlador del informe Mapa de Riesgos

No. de la tarea: 20	**No. de la HU:** 13
Nombre de la tarea: Implementar el controlador del informe Mapa de Riesgos	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Este controlador instancia el árbol y realiza el cálculo del impacto acumulado y repercutido.	

Tabla 76: Tarea de ingeniería Implementar la vista del informe Mapa de Riesgos.

No. de la tarea: 21	**No. de la HU:** 13
Nombre de la tarea: Implementar la vista del informe Mapa de Riesgos.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Esta vista muestra las amenazas por cada activo y los activos que afecta cada amenaza. Debe generar el informe en PDF. Guiarse por la descripción del informe en el libro Catálogo de elementos de la MAGERIT.	

Tabla 77: Tarea de ingeniería Gestionar Salvaguarda

No. de la tarea: 22	**No. de la HU:** 14
Nombre de la tarea: Gestionar Salvaguarda	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Implementar los Assert en la entidad Salvaguarda. Ejecutar el comando doctrine:generate:crud sobre la entidad Salvaguarda.	

Tabla 78: Tarea de ingeniería Aplicar estilos vista CRUD de Salvaguarda

No. de la tarea: 23	**No. de la HU:** 14
Nombre de la tarea: Aplicar estilos vista CRUD de Salvaguarda	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Aplicarle los estilos a las vistas del CRUD de Salvaguarda utilizando Bootstrap.	

Tabla 79: Tarea de ingeniería Calcular eficacia de un paquete de salvaguardas.

No. de la tarea: 24	**No. de la HU:** 15
Nombre de la tarea: Calcular eficacia de un paquete de salvaguardas.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5

Programador responsable: Roides Cruz Lara
Descripción: Se realiza la funcionalidad que permita calcular la eficacia de un conjunto de salvaguardas acumuladas sobre un activo.

Tabla 80: Tarea de ingeniería Aplicar una salvaguarda sobre un tipo de activo

No. de la tarea: 25	**No. de la HU:** 16
Nombre de la tarea: Aplicar una salvaguarda sobre un tipo de activo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se realiza la funcionalidad que permita proteger un tipo de activo con una salvaguarda.	

Tabla 81: Tarea de ingeniería Implementar el controlador del informe Evaluación de salvaguardas

No. de la tarea: 26	**No. de la HU:** 17
Nombre de la tarea: Implementar el controlador del informe Evaluación de salvaguardas	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Se cargan todas las salvaguardas y los riesgos que reducen.	

Tabla 82: Tarea de ingeniería Implementar el método para calcular riesgo acumulado.

No. de la tarea: 27	**No. de la HU:** 18
Nombre de la tarea: Implementar el método para calcular riesgo acumulado.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.2
Programador responsable: Roides Cruz Lara	
Descripción: Método que recorre el árbol desde el nodo afectado gasta sus hijos cumulado riesgos.	

Tabla 83: Tarea de ingeniería Implementar el método para calcular el riesgo repercutido

No. de la tarea: 28	**No. de la HU:** 19
Nombre de la tarea: Implementar el método para calcular el riesgo repercutido	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.2
Programador responsable: Roides Cruz Lara	
Descripción: Método que recorre el árbol desde el nodo afectado hasta sus superiores actualizando su riesgo repercutido.	

Tabla 84: Tarea de ingeniería Implementar el método para calcular el riesgo residual

No. de la tarea: 29	**No. de la HU:** 20
Nombre de la tarea: Implementar el método para calcular el riesgo residual	

Tipo de tarea: desarrollo	**Puntos estimados:** 0.2
Programador responsable: Roides Cruz Lara	
Descripción: Se cargan los paquetes de salvaguardas por cada activo y los riesgos que lo afectan calculando así los riesgos residuales. Ver Guía de Técnicas.	

Tabla 85: Tarea de ingeniería Implementar el controlador del informe Estado del Riesgo

No. de la tarea: 30	**No. de la HU:** 21
Nombre de la tarea: Implementar el controlador del informe Estado del Riesgo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se cargan todos los activos, valoraciones, riesgos y dependencias. Se instancia el árbol de activos y calculan los riesgos acumulados repercutido y su respectivo residual.	

Tabla 86: Tarea de ingeniería Implementar la vista del informe Estado del Riesgo

No. de la tarea: 31	**No. de la HU:** 22
Nombre de la tarea: Implementar la vista del informe Estado del Riesgo	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.5
Programador responsable: Roides Cruz Lara	
Descripción: Se muestran todos los activos con su categoría, sus riesgos potenciales, repercutidos y acumulados en la dimensión correspondiente. Debe mostrar graficas	

sencillas que apoyen la información anterior como una gráfica del riesgo total y total residual por tipos de activos.

Tabla 87: Tarea de ingeniería Gestionar Riesgos

No. de la tarea: 32	**No. de la HU:** 23
Nombre de la tarea: Gestionar Riesgos	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.10
Programador responsable: Roides Cruz Lara	
Descripción: Ejecutar el comando doctrine:generate:crud sobre la entidad Riesgo.	

Tabla 88: Tarea de ingeniería Implementar la vista de evaluación y tratamiento

No. de la tarea: 33	**No. de la HU:** 23
Nombre de la tarea: Implementar la vista de evaluación y tratamiento	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.10
Programador responsable: Roides Cruz Lara	
Descripción: La vista permite evaluar los riesgos y darles tratamiento (eliminarlos compartirlos o aceptarlos). Se debe aplicar los estilos css y funciones JavaScript usando el Bootstrap.	

Tabla 89: Tarea de ingeniería Gestionar Usuario

No. de la tarea: 34	**No. de la HU:** 24
Nombre de la tarea: Implementar las funcionalidades de crear, leer, actualizar y eliminar un usuario.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Instalar el Bundle FOS_User. Extender la clase User de la clase BaseUser de FOS UserBundle**.** Ejecutar el comando doctrine:generate:crud en la entidad User.	

Tabla 90: Tarea de ingeniería

No. de la tarea: 25	**No. de la HU:** 17
Nombre de la tarea: Aplicar estilos a las vistas del CRUD de User.	
Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Aplicar los estilos css y funciones JavaScript utilizando Bootstrap.	

Tabla 91: Tarea de ingeniería Autenticar usuario

No. de la tarea: 25	**No. de la HU:** 17
Nombre de la tarea: Autenticar usuario.	

Tipo de tarea: desarrollo	**Puntos estimados:** 0.25
Programador responsable: Roides Cruz Lara	
Descripción: Aplicar la configuración en el archivo config.yml para indicar el proveedor de usuarios del sistema, la ruta de login y el check.	

Anexo 4

Tabla 92: Caso de prueba de aceptación

Caso de prueba de aceptación Identificar los activos del inventario de GLPI	
Código: 1	**HU:** 1
Nombre: Identificar los activos del inventario de GLPI	
Descripción: prueba para la funcionalidad que permite identificar los activos reportados en GLPI.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y hacer clic en el botón Identificar activos de GLPI.	
Resultado esperado: El sistema muestra una tabla con los activos reportados en GLPI, su serial, su clasificación según la MAGERIT	
Evaluación de la prueba: satisfactoria	

Tabla 93: Caso de prueba de aceptación

Caso de prueba de aceptación Establecer dependencias entre los activos del inventario de GLPI	
Código: 2	**HU:** 2

Nombre: Establecer dependencias entre los activos del inventario de GLPI
Descripción: Prueba para la funcionalidad que permite establecer dependencias entre los activos del inventario de GLPI
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y hacer clic en un activo identificado.
Resultado esperado: El sistema muestra el activo reportado y una lista de dependencias o activos de los que depende directamente.
Evaluación de la prueba: satisfactoria

Tabla 94: Caso de prueba de aceptación

Caso de prueba de aceptación Gestionar activos.	
Código: 3	**HU:** 3
Nombre: Gestionar activos	
Descripción: Prueba funcionalidades permiten al usuario crear, editar, mostrar y eliminar todos los datos de un activo.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y dar clic en el botón Añadir activo. En caso de que desee editar, eliminar, actualizar o ver los datos de un activo hacer clic en un activo y elegir la funcionalidad correspondiente.	
Resultado esperado: El sistema permite al usuario crear, editar, mostrar y eliminar todos los datos de un activo.	
Evaluación de la prueba: satisfactoria	

Tabla 95: Caso de prueba de aceptación

Caso de prueba de aceptación Valorar activo.	
Código: 4	**HU:** 4
Nombre: Valorar activo	
Descripción: Prueba a la funcionalidad que permite valorar a los activos en las diferentes dimensiones de valoración	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y hacer clic en el botón valorar activos. Debe seleccionar la dimensión o escoger los criterios de valoración con los cumple el activo o activos a valorar.	
Resultado esperado: El sistema debe actualizar la tabla con los valores que el usuario seleccionó.	
Evaluación de la prueba: satisfactoria	

Tabla 96: Caso de prueba de aceptación

Caso de prueba de aceptación Valorar activo.	
Código: 5	**HU:** 5
Nombre: Establecer dependencias directas entre activos	
Descripción: Prueba a la funcionalidad que permite establecer dependencias directas entre activos.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y hacer clic sobre el nombre de un activo. Dar clic en el botón añadir activo.	

Resultado esperado: En la vista mostrar activo debe mostrarse los activos de los que depende directamente. Permitir que el usuario añada una o varias dependencias especificando un grado y una justificación. Los activos añadidos se mostraran en la tabla de dependencias.
Evaluación de la prueba: satisfactoria

Tabla 97: Caso de prueba de aceptación

Caso de prueba de aceptación Establecer dependencias indirectas entre activos.	
Código: 6	**HU:** 6
Nombre: Caso de prueba de aceptación establecer las dependencias indirectas entre activos	
Descripción: Prueba a la funcionalidad que establece las dependencias indirectas entre activos.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: Hacer clic en el botón Modelo del Valor.	
Resultado esperado: El sistema debe mostrar la tabla para cada activo sus activos superiores y sus hijos y el grado de dependencias.	
Evaluación de la prueba: satisfactoria	

Tabla 98: Caso de prueba de aceptación

Caso de prueba de aceptación Calcular valor acumulado.	
Código: 7	**HU:** 7
Nombre: Caso de prueba de aceptación Calcular valor acumulado.	
Descripción: Prueba a la funcionalidad que calcula el valor acumulado en un activo por	

sus superiores.
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.
Entradas/Pasos de ejecución: Hacer clic en el botón Modelo del Valor.
Resultado esperado: El sistema debe mostrar para cada activo su valor propio y acumulado.
Evaluación de la prueba: satisfactoria

Tabla 99: Caso de prueba de aceptación

Caso de prueba de aceptación Generar informe Modelo del valor.	
Código: 8	**HU:** 8
Nombre: Caso de prueba de aceptación Generar informe Modelo del valor.	
Descripción: Prueba a la funcionalidad Genera el informe Modelo del valor.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: Hacer clic en el botón Modelo del Valor. Hacer clic en botón generar informe.	
Resultado esperado: El sistema debe mostrar para cada activo su valor propio y acumulado, sus superiores e hijos. Debe exportar a PDF este informe.	
Evaluación de la prueba: satisfactoria	

Tabla 100: Caso de prueba de aceptación

Caso de prueba de aceptación Identificar amenazas de GLPI.	
Código: 9	**HU:** 9
Nombre: Caso de prueba de aceptación Identificar amenazas de GLPI.	
Descripción: Funcionalidad que permite identificar las amenazas de GLPI	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir Amenazas y hacer clic en el botón identificar amenazas de GLPI.	
Resultado esperado: El sistema debe actualizar la tabla con las amenazas reportadas en GLPI	
Evaluación de la prueba: satisfactoria	

Tabla 101: Caso de prueba de aceptación

Caso de prueba de aceptación Gestionar amenaza.	
Código: 10	**HU:** 10
Nombre: Caso de prueba de aceptación Gestionar amenaza	
Descripción: Funcionalidad que permite crear, editar, mostrar y eliminar todos los datos de una amenaza.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir al inventario de activos y dar clic en el botón Añadir amenaza. En caso de que desee editar, eliminar, actualizar o ver los datos de una amenaza debe hacer clic en un activo y elegir la funcionalidad correspondiente.	
Resultado esperado: El sistema permite al usuario crear, editar, mostrar y eliminar todos los datos de un activo.	

Evaluación de la prueba: satisfactoria

Tabla 102: Caso de prueba de aceptación

Caso de prueba de aceptación Relacionar amenazas con activos.	
Código: 11	**HU:** 11
Nombre: Relacionar las amenazas con los activos que afecta.	
Descripción: Caso de prueba de aceptación para la funcionalidad que permite relacionar las amenazas con los activos que afecta.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir a amenazas y hacer clic en el botón sobre una amenaza. Debe seleccionar el botón añadir activo. Debe especificar la frecuencia, la degradación y la dimensión que afecta.	
Resultado esperado: El sistema debe permitir crear un nuevo riesgo.	

Tabla 103: Caso de prueba de aceptación

Caso de prueba de aceptación Calcular impacto repercutido y acumulado de una amenaza.	
Código: 12	HU: 12
Nombre: Caso de prueba de aceptación Valorar activo	
Descripción: Prueba para la funcionalidad que calcula impacto repercutido y acumulado de una amenaza.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir a Mapa de riesgo.	
Resultado esperado: El sistema debe calcular el impacto repercutido y acumulado de una amenaza en los activos.	

Evaluación de la prueba: satisfactoria

Tabla 104: Caso de prueba de aceptación

Caso de prueba de aceptación Generar informe Mapa de Riesgos.	
Código: 13	**HU:** 13
Nombre: Caso de prueba de aceptación Generar informe Mapa de Riesgos.	
Descripción: Prueba para la funcionalidad que genera el informe Mapa de riesgos.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: Ir a informes y seleccionar Mapa de Riesgos. Seleccionar el botón Generar informe.	
Resultado esperado: El sistema debe generar un informe de las amenazas posibles, caracterizadas por su frecuencia de ocurrencia y la degradación que causarían en los activos. Debe permitir exportar a PDF.	
Evaluación de la prueba: satisfactoria	

Tabla 105: Caso de prueba de aceptación

Caso de prueba de aceptación Gestionar salvaguarda.	
Código: 14	**HU:** 14
Nombre: Caso de prueba de aceptación Gestionar salvaguarda.	
Descripción: Prueba para la funcionalidad que permiten crear, mostrar, actualizar y eliminar una salvaguarda.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El usuario debe ir salvaguardas y hacer clic en el botón	

añadir salvaguarda. En caso de que desee editar, eliminar, actualizar o ver los datos de un activo hacer clic en un activo y elegir la funcionalidad correspondiente.
Resultado esperado: El sistema debe permitir al administrador crear, editar, mostrar y eliminar todos los datos de una salvaguarda.
Evaluación de la prueba: satisfactoria

Tabla 106: Caso de prueba de aceptación

Caso de prueba de aceptación Calcular eficacia de un paquete de salvaguardas	
Código: 15	**HU:** 15
Nombre: Caso de prueba de aceptación Calcular eficacia de un paquete de salvaguardas	
Descripción: Prueba para la funcionalidad que calcula la eficacia de un paquete de salvaguardas sobre de un activo.	
Condiciones de ejecución: El usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: Ir a Evaluación de Salvaguardas.	
Resultado esperado: El sistema debe calcular la eficacia de un paquete de salvaguardas acumuladas sobre un activo, y mostrar a cada activo con las salvaguardas que lo protegen	
Evaluación de la prueba: satisfactoria	

Tabla 107: Caso de prueba de aceptación

Caso de prueba de aceptación Aplicar una salvaguarda.	
Código: 16	**HU:** 16
Nombre: Nombre: Caso de prueba de aceptación Aplicar una salvaguarda.	
Descripción: Prueba para la funcionalidad que aplicar una salvaguarda a un tipo de	

activo
Condiciones de ejecución: el usuario debe estar autenticado en el sistema.
Entradas/Pasos de ejecución: El usuario debe ir Salvaguardas seleccionar la categoría, ir a la salvaguarda que se quiere editar y seleccionar tipo de activo a proteger.
Resultado esperado: El sistema debe permitir al usuario aplicar una salvaguarda a un tipo de activo.
Evaluación de la prueba: satisfactoria

Tabla 108: Caso de prueba de aceptación

Caso de prueba de aceptación Generar informe Evaluación de salvaguardas.	
Código: 17	**HU:** 17
Nombre: Caso de prueba de aceptación Generar informe Evaluación de salvaguardas.	
Descripción: Prueba para la funcionalidad permite generar informe Evaluación de salvaguardas	
Condiciones de ejecución: el usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: Seleccionar el informe Evaluación de Salvaguardas. Hacer clic en el botón Generar informe.	
Resultado esperado: El sistema debe Generar informe Evaluación de salvaguardas desplegadas y caracterizadas por su grado de efectividad. Debe permitir exportar a PDF.	
Evaluación de la prueba: satisfactoria	

Tabla 109: Caso de prueba de aceptación

Caso de prueba de aceptación Generar informe Estado del Riesgo.	
Código: 18	**HU:** 18,19,20,21,22

Nombre: Caso de prueba de aceptación Generar informe Estado del Riesgo.
Descripción: Prueba para la funcionalidad que permite generar informe Estado del Riesgo.
Condiciones de ejecución: el usuario debe estar autenticado en el sistema.
Entradas/Pasos de ejecución: Seleccionar el informe Evaluación de Salvaguardas. Hacer clic en el botón Generar informe.
Resultado esperado: Informe que detalla para cada activo los riesgos acumulados, repercutidos, potenciales y residuales ocasionados por las amenazas.
Evaluación de la prueba: satisfactoria

Tabla 110: Caso de prueba de aceptación

Caso de prueba de aceptación Gestionar Riesgos.	
Código: 19	**HU:** 23
Nombre: Caso de prueba de aceptación Gestionar riesgo.	
Descripción: Prueba para la funcionalidad que permiten crear, mostrar, actualizar y eliminar un riesgo.	
Condiciones de ejecución: el usuario debe estar autenticado en el sistema.	
Entradas/Pasos de ejecución: El riesgo se crea cuando se relaciona una amenaza con un activo. Para editar la clasificación del riesgo ir a Evaluación, para tratarlo ir a Tratamiento.	
Resultado esperado: El sistema debe permitir evaluar el riesgo en Crítico, Grave, Apreciable y Asumible. Debe permitir realizar un tratamiento de los riesgos como: eliminar el riesgo eliminando sus causas: información tratada, servicios prestados, arquitectura del sistema, reducir o limitar el impacto, reducir la probabilidad de que la amenaza ocurra.	
Evaluación de la prueba: satisfactoria	

Tabla 111: Caso de prueba de aceptación

Caso de prueba de aceptación Autenticar usuario.	
Código: 20	**HU:** 24
Nombre: Caso de prueba de aceptación Autenticar usuario.	
Descripción: Prueba para la funcionalidad que permite autenticar un usuario en el sistema.	
Condiciones de ejecución:	
Entradas/Pasos de ejecución: Ir al formulario login.	
Resultado esperado: El sistema debe autenticar al usuario en el sistema y dar acceso en caso de que la operación resulte satisfactoria.	
Evaluación de la prueba: satisfactoria	

Tabla 112: Caso de prueba de aceptación

Gestionar usuario	
Código: 21	**HU:** 25
Nombre: Caso de prueba de aceptación Gestionar usuario	
Descripción: Prueba para la funcionalidad que permite crear, mostrar, actualizar, eliminar un usuario del sistema.	
Condiciones de ejecución: El usuario debe estar autenticado como Administrador en el sistema.	
Entradas/Pasos de ejecución: Seleccionar Administrar usuario. Para mostrar dar clic sobre el nombre del usuario. Las demás opciones se seleccionan en la vista de mostrar	

Resultado esperado: El sistema permite crear, mostrar, actualizar, eliminar un usuario del sistema.
Evaluación de la prueba: satisfactoria