Universidad de las Ciencias Informáticas

Facultad 2

Título: Agente de Auditoría de Seguridad Informática (AASI).

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Jeacny Reyes Echevarria

Tutores: Ing. María de los Ángeles Rubalcaba Betancourt

Ing. Yosbany Tejas de la Cruz

Ciudad de La Habana, Cuba

Junio, 2013

PENSAMIENTO

Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad.

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Jeacny Reyes Echevarria

_____

Ing. Yosbany Tejas de la Cruz

_____

Ing. María de los A. Rubalcaba Betancourt

_____

AGRADECIMIENTOS

Primeramente a mis padres por su apoyo incondicional, por estar presentes en los momentos difíciles, por sus consejos, por el amor que me profesan, por la educación que me han dado, en general, por todo, sin ellos no hubiese sido posible lograr esta meta, ni nada de lo que hasta ahora he logrado y soy, por ello les estoy eternamente agradecidos y orgulloso de la persona que son y de ser su hijo.

A mi familia por las relaciones reciprocas de amor, primero a mis abuelos por ser el pilar de los valores creados, el ejemplo a seguir y los culpables de mantenernos unidos. A mis tíos que son como mis padres, a mis primos que son los hermanos que no tengo, en general, a todos, porque han sido un elemento esencial en la formación de mi persona y de la estabilidad espiritual y moral que me ayudó a no desviarme nunca del camino. Los presentes físicamente y los que no, sé que están orgullosos de mí y del objetivo alcanzado.

A mis amigos de la Universidad, Camilo, Adrián, Chamizo, Dany, Bauza, Ariam (filio), en general a todos, pero esencialmente con los que he convivido en los últimos años y los quiero como mi familia, a Ramiro, Pichi, Julio, Rogelio, Carlo, Rosalis, Wendy, Adriana y Lili, con ellos he vivido momentos memorables y he podido contar siempre con su ayuda incondicional.

A todos los profesores que durante mi carrera estudiantil han influido en mi formación como profesional, a mis tutores con los cuales he podido contar durante todo el trabajo de diploma y en especial a la profesora Maura Berta, por la confianza que depositó en mí.

A la Revolución y la UCI por brindarme esta oportunidad.

DEDICATORIA

A Maribel Echevarría González y Francisco Reyes Graverán, mis padres. A toda mi familia, los presentes físicamente y los que no.

RESUMEN

En el presente trabajo se describe la construcción de una solución informática que actúe como Agente de Auditoría de Seguridad Informática (AASI) que permita la detección de las violaciones a las políticas de seguridad en el menor tiempo posible en el Centro de Informatización de la Seguridad Ciudadana (ISEC) de la Facultad 2 en la Universidad de las Ciencias Informáticas (UCI).

La solución se construye en un ambiente de escritorio, siguiendo la metodología ágil XP para su desarrollo, sin la presencia de otros sistemas precedentes, capaz de realizar la auditoría tanto automática como manualmente, informar al usuario las incidencias detectadas y guardar los informes de auditorías en la Base de Datos (BD) para su seguimiento. Una herramienta multiplataforma que inicia con el sistema operativo (SO) y se mantiene corriendo como un servicio.

Con el fin de lograr este propósito, se realiza un estudio de las auditorías de seguridad informática en el Centro ISEC. Se definen los requerimientos que debe cumplir la aplicación, modelándose además todo el proceso de construcción de la misma hasta la etapa de pruebas. Durante esta etapa se realizan las pruebas de unidad y las pruebas de aceptación, esta última con la participación del cliente como probador, obteniéndose resultados satisfactorios en ambas.

PALABRAS CLAVE: Auditoría, seguridad, informática, sistema operativo, agente, herramienta, historias de usuario, prueba.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La auditoría es una disciplina que se practica desde los inicios de la civilización. Desde muy temprano, el hombre se percató que, para evitar todo tipo de fraude en sus activos contables, era necesaria una correcta inspección de sus cuentas por personas especializadas y ajenas al proceso; que garantizaran los resultados sin sumarse o participar en el desfalco.

“En un principio sus objetivos primordiales eran la detección y prevención del fraude, aunque posteriormente hubo un cambio decisivo en la demanda y el servicio, teniendo como propósito principal cerciorarse de la condición financiera actual y de las ganancias de una empresa” (1). Con el paso del tiempo y la aparición de las grandes empresas, el campo de acción de la auditoría se ha visto en la necesidad de ampliarse, surgiendo diferentes tipos de auditorías como son la auditoría fiscal, operacional, financiera, medioambiental, y con el desarrollo de las Tecnologías de la Información (TIC) la auditoría informática.

La auditoría informática es de vital importancia para el buen desempeño de los sistemas de información, proporciona los controles necesarios para que los sistemas sean confiables y con un buen nivel de seguridad. “Consiste en recoger, agrupar y evaluar evidencias para determinar si un sistema informático salvaguarda el activo empresarial, mantiene la integridad de los datos, lleva a cabo los fines de la organización, utiliza eficientemente los recursos, y cumple con las leyes y regulaciones establecidas.” (2)

Este proceso brinda muchos beneficios para las empresas y sistemas de información garantizando confianza en los usuarios sobre la seguridad y control de los servicios que prestan, mejorando las relaciones internas y del clima de trabajo, disminuyendo sus costos, mejorando su imagen y características como desempeño, fiabilidad, eficacia, rentabilidad, privacidad, y seguridad.

Para lograr estas características existen diferentes tipos de auditorías informáticas, enfocadas a mejorar las distintas esferas que se definen dentro de los sistemas informáticos como la auditoría de sistemas, datos, seguridad, seguridad física y seguridad lógica. La auditoría de sistemas se realiza para mejorar el desempeño del software de sistemas y el de aplicación; la auditoría de datos para la clasificación de los datos, estudio de las aplicaciones y análisis de los flujogramas; la auditoría de seguridad para verificar disponibilidad, integridad, confidencialidad, autenticación y no repudio en datos e información; la de seguridad física se refiere a la seguridad del hardware y los soportes de datos, así como la protección externa y del entorno, y la de seguridad lógica a la seguridad en el uso de software, la protección de los

datos, procesos y programas, así como la del acceso ordenado y autorizado de los usuarios a la información.

Para detectar evidencias de riesgos y fallas de seguridad en los procesos de negocios originados por un mal uso en los sistemas informáticos se realiza la auditoría de seguridad informática, la misma analiza los procesos relacionados únicamente con la seguridad, esta puede ser física o lógica pero siempre orientada a la protección de la información. “Consiste en analizar el nivel de seguridad de nuestro sistema informático utilizando metodologías y herramientas para conocer cuáles son los problemas a los que nos podemos enfrentar, presentarlos en un informe y proponer las medidas que sería necesario aplicar para solucionarlos.” (3)

Las metodologías deben cumplir con un conjunto de etapas de vital importancia para la realización exitosa de una auditoría de seguridad informática, estas etapas son: definir el alcance y objetivos de la auditoría informática, realizar un estudio inicial del entorno auditable, determinar los recursos necesarios para realizar la auditoría, elaborar un plan y programas de trabajo, realizar las actividades propiamente dichas de la auditoría, y por último confeccionar y elaborar un informe final.

Las herramientas realizan un conjunto de acciones, enfocadas en asegurar una mayor integridad, confidencialidad y confiabilidad de la información; permitiendo conocer la situación actual del área informática, lograr la seguridad de los datos, hardware, software e instalaciones y minimizar la existencia de riesgos en el uso de la tecnología de la información. Son utilizadas para prevenir amenazas o riesgos como la intercepción de las comunicaciones, acceso no autorizado a ordenadores o redes de ordenadores, programas que le modifiquen o dañen los datos y robo e infiltración de datos críticos. Permiten identificar detalladamente los accidentes o acciones malintencionadas, que pongan en peligro la disponibilidad, autenticidad, integridad y confidencialidad de los datos almacenados.

Entre las herramientas usadas como apoyo para los procesos de auditoría de seguridad informática, se encuentran las que utilizan la tecnología de agentes. Una aplicación para auditorías de seguridad informática con tecnología de agentes permite auditar de forma automática los sistemas informáticos sin la necesidad de la presencia de auditores, los cuales solo interactúan con el agente para activarlo y especificarle la información que desean recoger o bien para observar los resultados obtenidos de la ejecución de las tareas agendadas.

La auditoría de seguridad informática ofrece significativos beneficios en las empresas y sistemas informáticos proporcionando un aumento en la motivación del personal y de su compromiso con la misión de la compañía, mejoras de las relaciones laborales, del rendimiento, del profesionalismo, y del clima laboral de los recursos humanos (RRHH), además ayuda a formar equipos competentes e impulsar un aumento de la productividad de la empresa. (3)

El Centro ISEC es una entidad desarrolladora de software perteneciente a la facultad 2 en la Universidad de las Ciencias Informáticas (UCI), que tiene la necesidad de realizar auditorías de seguridad informática con el objetivo de identificar, enumerar y describir las vulnerabilidades que constituyen amenazas para la seguridad de la información en el Centro.

En los inicios del Centro ISEC, para enero de 2010, se hacía difícil identificar las fallas de seguridad ya que la Universidad y el Centro no contaban con una herramienta que ayudara a los auditores a facilitar el proceso de auditoría, este proceso se realizaba de forma manual anotando en un excel el resultado de comprobar si cada una de las computadoras del centro cumplía con las políticas de seguridad descritas en la Guía de Control de la Seguridad Informática (GCSI) de ISEC.

En la actualidad, el Centro cuenta con una aplicación web llamada Sistema de Gestión de la Seguridad Informática (SIGESI) que gestiona los procesos de auditorías. En SIGESI se introduce el resultado de comprobar las políticas de seguridad en las computadoras del Centro, la misma emite reportes al asesor de auditoría informándole de los elementos que constituyen amenazas de seguridad. Vale destacar que esta aplicación no ha facilitado mucho el proceso de auditorías ya que continúan realizándose de forma manual e introduciendo después dentro de la web la información recogida, lo que pudiera incurrir en errores durante el proceso de captura de datos significativos y en el posterior traslado a la aplicación.

Las auditorías se realizan con una frecuencia aproximada de una semana, nunca inferior, en la que se auditan una muestra de diez computadoras del Centro, escogidas de forma aleatoria; también se realiza, una vez por mes, una auditoría general a todos los ordenadores pertenecientes al Centro; existiendo un espacio de tiempo significativo durante el cual no son comprobadas las políticas de seguridad en la mayoría de las computadoras del Centro, quedando las que no cumplan expuestas a posibles ataques o acciones malintencionadas ya que no son identificadas durante dicho tiempo las violaciones a las políticas de seguridad informática.

Además de esto se suma el inconveniente de que el Centro no cuenta, para detectar las incidencias de seguridad en sus ordenadores, con la existencia de una herramienta de apoyo, que automatice el proceso de auditoría de seguridad informática. La principal dificultad que surge en el proceso de auditoría de seguridad informática en ISEC es que debido a lo anteriormente expuesto se incurre en demoras de tiempo para identificar las vulnerabilidades que constituyen amenazas reales para la seguridad de la información.

Dada la situación problemática anteriormente planteada el **problema a resolver** es: ¿Cómo detectar a tiempo las incidencias de seguridad informática en el Centro ISEC de la Facultad 2 en la UCI que permita facilitar y mejorar el proceso de auditoría de seguridad informática en el Centro?

Se define como **objeto de estudio:** Auditorías de Seguridad Informática.

El **objetivo general** es: Desarrollar una herramienta que actúe como Agente de Auditoría de Seguridad Informática (AASI) para el Centro ISEC que permita la detección de incidencias de seguridad informática en el menor tiempo posible.

Como **campo de acción** se determina: Agente de Auditoría de Seguridad Informática para el Centro ISEC de la Facultad 2 en la UCI.

Los **objetivos específicos** que se derivan son:

1. Diseñar una solución de la herramienta propuesta para el Centro ISEC.
2. Implementar la herramienta AASI para el Centro ISEC.
3. Validar la herramienta informática propuesta.

Las **tareas científicas** a desarrollar para dar cumplimiento a los objetivos trazados son:

1. Revisión bibliográfica de los fundamentos teóricos referentes a seguridad informática, auditorías informáticas y auditorías de seguridad informáticas para lograr una mayor comprensión en este tema.
2. Evaluación de las herramientas existentes para la realización de auditorías de seguridad informática con el fin de conocer si le darían solución a los problemas de seguridad del Centro.
3. Selección de la metodología, herramientas y tecnologías que mejor se ajusten para la creación de un Agente de Auditorías de Seguridad Informática.

4. Selección de la arquitectura a emplear para la construcción de un Agente de Auditorías de Seguridad Informática.
5. Implementación de un Agente de Auditorías de Seguridad Informática, de forma tal, que permita arribar a un resultado exitoso en la detección de las violaciones de seguridad informática en el Centro ISEC.
6. Validación del Agente de Auditorías de Seguridad Informática para verificar que cumpla con las necesidades y expectativas del Centro.

Para la realización de la investigación se aplicó el método científico teórico, el cual facilitó la recopilación de la información necesaria para la modelación del diseño de la solución propuesta. El método teórico utilizado fue el analítico-sintético, se define con el objetivo de modelar el funcionamiento de los procesos de auditorías de seguridad informática, así como obtener, resumir y describir los elementos más importantes relacionados con estos procesos para construir el agente de auditorías de seguridad informática propuesto.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se abordan varios aspectos teóricos, conceptos y definiciones relacionados con el tema que se desarrolla en este trabajo, con el propósito de tener una base teórica para la construcción de la herramienta. Se profundiza en los conceptos de agente, auditoría, seguridad informática, auditoría informática y auditoría de Seguridad Informática. Además de puntualizar sobre los diferentes tipos de auditorías informáticas, auditorías en ISEC, tecnología de agentes, herramientas existentes y metodologías de desarrollo de software.

1.1 Marco conceptual

A continuación se detallan los conceptos relacionados con el entorno del problema para un mayor entendimiento del marco teórico elaborado.

Agente: Un agente es un programa de ordenador que actúa autónomamente en nombre de una persona u organización, lleva a cabo una serie de operaciones en nombre de un usuario o de otro programa, con algún grado de independencia o autonomía, empleando algún conocimiento o representación de los objetivos o deseos del usuario (4).

Auditoría: La auditoría es un examen crítico pero no mecánico, que no implica la preexistencia de fallos en la entidad auditada y que persigue el fin de evaluar y mejorar la eficacia y eficiencia de una sección o de un organismo. (5)

Seguridad Informática: La seguridad informática consiste en aquellas prácticas que se llevan adelante respecto de un determinado sistema de computación a fin de proteger y resguardar su funcionamiento y la información en él contenida para garantizar su confidencialidad, integridad y disponibilidad. (6)

Auditoría Informática: Conjunto de procedimientos y técnicas para evaluar y controlar, total o parcialmente, un sistema informático, con el fin de proteger sus activos y recursos, verificar si sus actividades se desarrollan eficientemente y de acuerdo con la normativa informática y general existentes en cada empresa para conseguir la eficacia exigida en el marco de la organización correspondiente. (7)

Auditoría de seguridad Informática: La auditoría de seguridad informática consiste en el análisis crítico de los procesos relacionados con la seguridad de un sistema informático, ésta puede ser física o lógica pero siempre orientada a la protección de la información. (8)

1.2 Tipos de Auditorías informáticas

La auditoría informática vela por la correcta utilización de los recursos que una empresa dispone con el fin de garantizar un adecuado sistema de información, es la encargada de verificar que sus sistemas y procesos informáticos funcionen adecuadamente para las funciones que han sido programados y sus activos digitales se encuentren debidamente protegidos, distinguiéndose cinco tipos de auditoría informática:

Auditoría informática de sistemas: Revisa los sistemas de información actuales, tanto a nivel de hardware como de software, con el objeto de descubrir potenciales puntos de mejora y determinar en qué modo debería actuarse para mejorar tanto éstos como otros aspectos.

Auditoría informática de explotación: Revisa todos los procesos encargados de producir resultados, entre ellos la captura de la información, los sistemas de hardware de explotación, los RRHH de explotación y otros.

Auditoría informática de comunicaciones: Revisa la topología de red y determina posibles mejoras, analiza caudales y grados de utilización.

Auditoría informática de desarrollo: Revisa el proceso completo de desarrollo de proyectos por parte de la empresa auditada. El análisis se basa en cuatro aspectos fundamentales: revisión de las metodologías utilizadas, control interno de las aplicaciones, satisfacción de los usuarios y control de procesos y ejecuciones de programas críticos.

Auditoría informática de seguridad: Abarca los conceptos de seguridad física y seguridad lógica. La seguridad física se refiere a la protección del hardware y de los soportes de datos, así como a la de los edificios e instalaciones que los albergan y la seguridad lógica asociada a la protección del software, así como el acceso ordenado y autorizado de los usuarios a la información.

1.2.1 Auditorías de seguridad informáticas en las empresas u organizaciones

“Las organizaciones, sus redes y sistemas de información, se enfrentan en forma creciente con amenazas relativas a la seguridad, incluyendo el fraude asistido por computadora, espionaje, sabotaje, vandalismo, incendio e inundación. Daños tales como los ataques mediante virus informáticos, "hacking" y denegación de servicio se han vuelto más comunes, ambiciosos y crecientemente sofisticados. La dependencia de las

organizaciones respecto de los sistemas y servicios de información denota que ellas son más vulnerables a las amenazas concernientes a seguridad.” (9)

La seguridad que puede lograrse por medios técnicos es limitada y debe ser respaldada por una gestión y procedimientos adecuados. La identificación de los controles que deben implementarse requiere una cuidadosa planificación y atención a todos los detalles.

La auditoría de seguridad informática en las empresas u organizaciones permite conocer en el momento de su realización cuál es la situación exacta de sus activos de información en cuanto a protección, control y seguridad. La misma consiste en realizar una evaluación de los sistemas informáticos pertenecientes a dichas empresas u organizaciones, con el fin de detectar errores y fallas; estos son plasmados en un informe bien detallado, en el que se describe. (10)

- Equipos instalados, servidores, programas y sistemas operativos.
- Análisis de seguridad en los equipos y en la red.
- Gestión de los sistemas instalados.
- Vulnerabilidades que pudieran presentarse en una revisión de las estaciones de trabajo, redes de comunicaciones, servidores.
- Protocolo de seguridad ante una amenaza tecnológica.

Una vez obtenidos los resultados y verificados, se emite otro informe, en el que se proponen soluciones a los problemas de seguridad detectados durante el proceso de auditoría, que les permitan a los administradores mejorar la seguridad de los sistemas informáticos de la organización o empresa a la que pertenecen.

Existen muchas metodologías para la realización de una auditoría de seguridad informática como OCTAVE, MAGERIT, CORAS o el Manual de código abierto para la realización de pruebas de seguridad (OSSTMM), publicado por el Instituto Superior de Empresa y Comunicación (ISECOM), que cumplen con los estándares a nivel internacional orientados a servir como base para auditorías informáticas, entre los estándares se encuentra COBIT (Objetivos de Control de la Tecnologías de la Información), la Organización Internacional de Normalización (ISO) responsable de la estandarización de normas relativas a las Tecnologías de la Información y las Comunicaciones (TICs), entre otros.

En nuestro país la seguridad informática está amparada legalmente por la Resolución 127 del 2007 del Ministerio de la Informática y las Comunicaciones, independientemente de su actividad rectora, no es suficiente para implementar una eficiente seguridad pues no propone, normas ni procedimientos a seguir para poder evaluar la misma, tan propensa en esos momentos a hechos delictivos o de corrupción, además se comprobó que es muy difícil evaluar la misma de forma más integral sin un documento que sirva de ayuda y guía para aquella persona que no sea especialista, por su amplitud y especificidades técnicas, para así poder prever a tiempo cualquier situación delictiva. (11)

En sentido general en las auditorías de seguridad informática se hace, según las necesidades de uso y dentro de las directivas de la organización o empresa; un análisis de vulnerabilidades, para controlar posibles fallos en la seguridad de los sistemas; se verifica el cumplimiento de la normativa y requisitos legales vigentes en materia de protección de datos personales para asegurar la confidencialidad y se comprueba el cumplimiento de la política de seguridad establecida para afirmar la integridad del sistema.

1.2.2 Auditorías informáticas en ISEC

En los inicios del Centro ISEC en enero de 2010, para realizarse una auditoría de seguridad informática, el asesor de auditoría de seguridad informática del Centro debía conformar un equipo de auditores. Este grupo de trabajo, de forma manual, anotaba en un excel el resultado de comprobar si cada una de las computadoras del Centro cumplía con las políticas de seguridad descritas en la GCSI. Posteriormente el asesor de auditoría de seguridad informática identificaba los problemas de seguridad existentes y los plasmaba en un informe, además se encargaba de realizar las acciones para erradicar las vulnerabilidades detectadas.

En muchas ocasiones no había personal disponible o capacitado para conformar el grupo de auditores, por lo que el asesor hacía todo el proceso de auditoría, desde la recogida de los elementos significativos hasta la emisión del reporte final y de las acciones correctivas para la eliminación de las fallas identificadas.

Actualmente las auditorías se realizan con una frecuencia aproximada de una semana, nunca inferior, en la que se auditan una muestra de diez computadoras del Centro escogidas de forma aleatoria; también se realiza, una vez por mes, una auditoría general a todos los sistemas infomáticos pertenecientes al Centro; el cual cuenta con una aplicación web llamada SIGESI que se encarga de la gestión del proceso de auditoría.

En SIGESI se introduce de forma manual el resultado de comprobar las políticas de seguridad de los sistemas informáticos pertenecientes al Centro, estos resultados son gestionados en la web donde se identifican las fallas e incidencias de seguridad encontradas en la auditoría realizada y se encarga de emitir un reporte al asesor de auditoría de seguridad informática con los problemas detectados.

1.3 Tecnología de agentes

Un agente es un programa capaz de controlar su proceso de toma de decisiones y de actuar, basado en la percepción de su ambiente, en persecución de uno o varios objetivos. El usuario "delega" en el agente una o varias tareas que debe llevar a cabo quedando a la espera de los resultados. (4)

Entre las características que diferencian estas herramientas de software de las demás, se encuentran: la Autonomía, ya que puede actuar sin la necesidad directa del hombre o de otro programa; la Continuidad temporal, siendo un proceso continuo el cual debe ejecutarse hasta que se haya alcanzado con el conjunto de objetivos solicitados o el usuario desee detenerlo y la Flexibilidad dado que sus acciones no están prefijadas permitiendo identificar el entorno en el que se está ejecutando y así definir las acciones a realizar para cumplir con los objetivos que le hayan sido solicitados. (12)

La tecnología de agentes distribuye en cada uno de los ordenadores pertenecientes a una subred una aplicación agente, que actúa de manera individual sin la necesidad de interactuar con el usuario, este solo lo activa y le especifica la información que desea obtener; una vez iniciado el agente se encarga de enviar dicha información, en caso de existir a un servidor central, sino la almacena localmente.

La tecnología de agente proporciona los siguientes beneficios (13):

- **Alta velocidad:** Analiza cientos o miles de equipos en unos pocos minutos.
- **Automatización:** Los agentes actualizan en el servidor el estado del cliente según una programación regular. Cada vez que se abre la aplicación, los usuarios pueden analizar una vista completa y al día de la seguridad de la red.
- **Precisión:** Los análisis locales tienen menos puntos de fallo que los remotos; los agentes continuarán funcionando incluso cuando los equipos no estén conectados a la red.

Un Agente de Auditoría de Seguridad Informática permite realizar auditorías automáticas en los ordenadores que se encuentra activado, ahorrándole este proceso a los auditores de seguridad

informática, los cuales podrán observar la información que brinda el agente, en caso de existir en un servidor central, sino en los computadores en que está activado dicho agente.

1.4 Herramientas relacionadas

Con el fin de detectar evidencias de riesgos y problemas en el apoyo informático a los procesos de negocio originados por un mal uso informático y del control, existe un conjunto de herramientas para auditorías de seguridad informática, que detectan los problemas de seguridad informática y le dan soluciones a muchos de dichos problemas. A continuación se describen algunas de estas herramientas.

1.4.1 Microsoft Baseline Security Analyzer (MBSA)

MBSA es una herramienta de fácil uso que ayuda a las pequeñas y medianas empresas a determinar su estado de seguridad de acuerdo con las recomendaciones de seguridad de Microsoft y ofrece orientación precisa sobre soluciones. Detecta errores de configuración de seguridad e identifica actualizaciones que faltan en los sistemas informáticos.

“Basado en la infraestructura de Windows Update Agent (Agente de Actualización de Windows) y Microsoft Update (Actualización de Microsoft). MBSA asegura coherencia con otros productos de administración de Microsoft, incluidos Microsoft Update, Windows Server Update Services (Servicios de Actualización de Servidores de Windows), Systems Management Server (Administración del Servidor Sistemas), System Center Configuration Manager (Manejador de Configuración del Centro del Sistema) y Small Business Server (Servidor de Negocio Pequeño).” (14)

Es ventajoso el uso de MBSA en un entorno empresarial de pequeña y mediana empresa, pero solo donde se trabaje con SO Windows ya que no soporta multi-plataforma; por lo que no daría solución a la situación problemática planteada ya que los sistemas informáticos pertenecientes al Centro ISEC utilizan los sistemas operativos Windows y Linux.

1.4.2 Nessus

“Nessus es un escáner de vulnerabilidades que funciona mediante un proceso de alta velocidad por el que encuentra los datos sensibles y trabaja con la auditoría de configuraciones y el perfil activo. Es de gran utilidad para los dispositivos de uso personal pero también para las grandes empresas que se manejan con equipos conectados en red, pues trabaja mediante el uso de DMZ, capaz de auditar y analizar tanto

los sistemas en red como los individuales a través del rastreo de archivos y parches de seguridad se adapta a diferentes plataformas y sistemas operativos.” (15)

Opcionalmente, los resultados del escaneo pueden ser exportados como informes en varios formatos, como archivos en texto plano, XML y HTML. Los resultados también pueden ser guardados en una base de conocimiento para referencia en futuros escaneos de vulnerabilidades.

Esta aplicación no constituye una solución a la situación problemática planteada puesto que no permite comprobar las políticas de seguridad informática específicas del Centro ISEC, además no implementa la tecnología de agentes.

1.4.3 Open Computer and Software Inventory (OCS Inventory)

OCS Inventory es una aplicación para realizar inventarios en los activos contables de las tecnologías de la información asociados a una red local, este procedimiento se realiza por medio de una estructura cliente servidor. Es un sistema multiplataforma que en un servidor recopila la información que le envía un software agente instalado en cada uno de los dispositivos de la subred, software con licencia GPL, libre de usar y copiar que permite además el despliegue de paquetes en computadores Windows y Linux.

“Se basa en los estándares actuales. El diálogo entre los equipos cliente y servidor se basa en el Protocolo de Transferencia de Hipertexto (HTTP) y el formato de los datos es XML. El servidor de administración utiliza Apache, MySQL y Perl, contiene 4 componentes principales: el servidor de base de datos, la comunicación con el servidor, el despliegue del servidor y la consola de administración. Tiene una interfaz web privativa escrita en el lenguaje de programación PHP que ofrece servicios complementarios como consulta del inventario, gestión de los derechos de los usuarios y una interfaz de servicio de ayuda para los técnicos.” (16)

La herramienta es ideal para trabajar con computadoras en red, trae muchos beneficios a los sistemas informáticos pertenecientes a una subred ya que recopila información sobre el hardware y software de equipos remotos y permite instalar aplicaciones o actualizaciones remotamente.

Con la herramienta sólo se podrá administrar el inventario de los archivos y no se podrá identificar las violaciones a las políticas de seguridad específicas del Centro ISEC por lo que no constituye una solución a la problemática anteriormente planteada.

1.4.4 GFI LANguard Network Security Scanner

GFI LANguard es una herramienta que detecta automáticamente debilidades de seguridad en la red. Proporciona total visibilidad del estado de la red mediante modernos informes sobre el nivel de service pack (en español paquete de servicio) de la máquina, parches de seguridad no instalados, recursos compartidos, puertos abiertos, servicios/aplicaciones activas en la máquina, entrada de claves de registro, contraseñas débiles, usuarios/grupos, y otros. Los resultados de la búsqueda se muestran en un informe HTML, que puede personalizarse, permitiendo asegurar la red proactivamente.

“Se integra con más de 1.500 aplicaciones críticas de seguridad de diversas categorías como: antivirus, cortafuegos, anti-phishing, clientes de redes privadas virtuales (VPN), filtrado de URL, gestión de parches, navegadores web, peer-to-peer, encriptación de disco, y otras. Se puede configurar para ejecutar en modo sin agente o con agente. La tecnología de agentes permite enviar la información recogida a un servidor central y distribuir la carga del análisis entre los equipos cliente. El administrador simplemente necesita definir el perímetro de red y proporcionar credenciales para habilitar el descubrimiento automático de red, el despliegue del agente y la auditoría de los equipos cliente. Sólo es necesaria la intervención manual cuando se requiere un ajuste.” (13)

GFI LANguard soporta multi-plataforma y realiza un conjunto de acciones que resultan beneficiosas para garantizar la seguridad de la red como son:

- Comprueba la falta de parches de seguridad y service pack.
- Implanta service pack y parches.
- Comprueba alertas/debilidades de seguridad.
- Detecta recursos compartidos y puertos abiertos innecesarios.
- Comprueba cuenta de usuario no utilizadas en las estaciones de trabajo.
- Comprueba la política de contraseñas y su fortaleza.

Esta herramienta no constituye una solución para el problema a resolver ya que no permite identificar las violaciones a las políticas de seguridad informática específicas del Centro ISEC, además es un software privativo, lo que implicaría que el Centro tuviese que pagar una licencia que permita su uso.

1.5 Metodologías de desarrollo de software

Las metodologías de desarrollo de software surgen con el objetivo de minimizar el tiempo de desarrollo de un software, hacen muy eficiente y reproducible el camino para obtener resultados confiables, contienen procedimientos de gestión que coordinan y guían técnicas, que determinan las herramientas necesarias para garantizar un eficaz soporte a automatizar.

Existen dos enfoques para las metodologías: ágiles y tradicionales. Las metodologías ágiles son lo más flexibles posibles, puesto que son utilizadas en proyectos de corta duración, generan poca documentación, y poseen además la característica de que el cliente forma parte del equipo de desarrollo; por su parte las metodologías tradicionales son usadas en grandes proyectos donde se requiera una gran cantidad de documentación durante todo el ciclo de vida del proyecto y en muchas ocasiones no se tiene un contacto directo con el cliente, el mismo no va a formar parte del equipo de trabajo.

1.5.1 Metodologías Tradicionales

Dan nombre a las primeras metodologías que se utilizaron para desarrollar software y que actualmente aún gozan de gran popularidad. A pesar de que el modelo en cascada fue a lo largo del siglo pasado la principal representante de este grupo de metodologías, actualmente podemos considerar el modelo de Proceso Unificado de Desarrollo (Rational Unified Process RUP por sus siglas en inglés) como su principal valedora. Se caracterizan por realizar un tratamiento predictivo de los proyectos y medir el progreso en términos de artefactos entregados, así como de la especificación de los requisitos, los documentos de diseño, planes de pruebas y revisiones de código.

1.5.1.1 Proceso Unificado de Desarrollo

RUP brinda al software un trabajo continuo en las disciplinas de asignación de responsabilidades en una organización de desarrollo. Se encarga de asegurar la construcción de un software con alta calidad, así como planificar, controlar e integrar todos los aspectos a tener en cuenta en el ciclo de vida del software.

Es una metodología tradicional que va más allá del análisis y diseño orientado a objetos para proporcionar un conjunto de técnicas de soporte, utiliza el lenguaje de modelado UML para describir un sistema. Establece y conserva modelos en lugar de enfocarse en la generación de gran cantidad de documentación.

Define cuatro elementos fundamentales:

- Trabajadores (¿Quién?)
- Actividades (¿Cómo?)
- Artefactos (¿Qué?)
- Flujo de actividades (¿Cuándo?)

En RUP se especifican 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería (Modelación de Negocio, Requerimientos, Análisis y diseño, Implementación, Prueba, Despliegue) y los tres restantes como flujos de apoyo (Administración del proyecto, Administración de configuración y cambios, y Ambiente).

1.5.2 Metodologías Ágiles

“Los procesos ágiles de desarrollo de software o anteriormente conocidos como metodologías livianas o ligeras, surgieron como una reacción en contra de los métodos pesados. En este sentido las metodologías ágiles estuvieron motivadas por una conciencia particularmente aguda de la crisis del software, por la responsabilidad que se le ha asignado a las grandes metodologías en la gestación de esa crisis y por el propósito de articular soluciones” (17).

“El término ágil fue aplicado a la industria del software en febrero del 2001 tras una reunión realizada en Utah, Estados Unidos, en la que se creó The Agile Alliance (La alianza ágil). Esta es una organización dedicada a promover los conceptos relacionados con el desarrollo ágil. El punto de partida que resume la filosofía ágil se encuentra en un documento denominado Agile Manifesto (Manifiesto ágil). En este documento se describe la filosofía a través de un conjunto de principios y valores” (18).

1.5.2.1 Principios expuestos en el manifiesto ágil

- La prioridad más alta es satisfacer al cliente a través de la entrega temprana y continua de software valioso.
- El cambio en los requisitos es bienvenido, incluso cuando llegan tarde en el desarrollo. Los procesos ágiles se acogen al cambio para lograr una ventaja competitiva para el cliente.
- Entregar software que funcione con frecuencia, desde un par de semanas hasta un par de meses, con preferencia en períodos de tiempo más cortos.
- La gente de negocios y los desarrolladores deben trabajar juntos regularmente a través de todo el proyecto.

- Construir proyectos en torno de individuos motivados. Darles la oportunidad y el respaldo que necesitan y darles confianza para que realicen sus tareas.
- La forma más eficiente y efectiva de comunicar información de un lado a otro dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la medida primaria de progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante indefinidamente.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad (el arte de maximizar la cantidad de trabajo que no se hace) es esencial.
- Las mejores arquitecturas, requerimientos y diseños emergen de equipos que se auto-organizan.
- A intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo, y de acuerdo a esto, ajusta su conducta. (18)

Las metodologías ágiles ofrecen una mayor importancia a las personas en vez de a los procesos y se caracterizan principalmente por el uso de técnicas para agilizar el desarrollo del software, así como de una mayor flexibilidad para adaptarse a los cambios en los requisitos del proyecto.

Entre las metodologías ágiles se encuentran:

1. Agile Project Management.
2. Crystal Methods.
3. Dynamic Sytems Development Method.
4. Scrum.
5. Test Driven Development.
6. Extreme Programming (XP).

1.5.3 Programación extrema (XP)

La programación extrema se caracteriza por ser una metodología ágil que facilita a los desarrolladores expertos la respuesta a las necesidades expuestas a cambios de los clientes, incluso a finales del ciclo de vida, es un proceso ligero, ágil, flexible; está orientado hacia quien produce y usa el software. Se

encuentra especialmente bien documentada con innumerables recursos online disponibles, comunidades libres y grupos de noticias y encontrándose una gran cantidad de proyectos realizados con dicha tecnología.

Formulada en 1999 por Kent Beck, la programación extrema es una forma ligera, eficiente, de bajo riesgo, flexible, previsible, científica y divertida de desarrollar software. Esta se distingue de las otras metodologías principalmente por (18):

- Su pronta, concreta y continua retroalimentación en ciclos cortos de tiempo.
- Su enfoque de planificación incremental, el cual rápidamente llega hasta un plan general que se espera que evolucione a través de la vida del proyecto.
- Su flexibilidad de flexibilizar la implementación de funcionalidades, respondiendo a los cambios que el negocio necesita.
- Su dependencia de las pruebas automatizadas escritas por los propios programadores y la supervisión por parte del cliente en el proceso de desarrollo para permitir la evolución del sistema y capturar los defectos antes de tiempo.
- Su dependencia de la comunicación oral, las pruebas y el código fuente para comunicar la intención y la estructura del sistema.
- Su dependencia de un proceso de diseño evolutivo que dura todo el tiempo en que se desarrolla el sistema.
- Su estrecha y cercana colaboración de los programadores con habilidades ordinarias.

Como toda metodología, XP busca mejorar la relación entre costo, tiempo, calidad y alcance del proyecto. Esto lo hace mediante tres objetivos:

- Facilitar los cambios.
- Gestionar los errores.
- Fomentar la colaboración entre el cliente y los desarrolladores.

Para lograr tales objetivos, Kent Beck propone prácticas de trabajo las cuales están condicionadas por los denominados valores.

1.5.3.1 Prácticas en XP

- **El juego de la planificación**: Es un permanente diálogo entre el cliente y los desarrolladores para lograr determinar rápidamente el alcance del proyecto.
- **Pequeñas entregas**: Colocan un sistema sencillo en producción que se actualiza de forma rápida y constante permitiendo que el verdadero valor de negocio del producto sea evaluado en un ambiente real. Estas entregas no pueden pasar las 2 o 3 semanas como máximo.
- **Metáfora**: Una metáfora es una historia que todo el mundo puede contar acerca de cómo funciona el sistema. Desarrollada por los programadores al inicio del proyecto, define una historia de cómo funciona el sistema completo. XP estimula historias, que son breves descripciones de un trabajo de un sistema en lugar de los tradicionales diagramas y modelos UML. La metáfora expresa la visión evolutiva del proyecto que define el alcance y propósito del sistema.
- **Diseño sencillo**: Se basa en la filosofía de que el mayor valor de negocio es entregado por el programa más sencillo que cumpla los requerimientos. Un diseño sencillo se enfoca en proporcionar un sistema que cubra las necesidades inmediatas del cliente, ni más ni menos. Este proceso permite eliminar redundancias y rejuvenecer los diseños obsoletos de forma sencilla.
- **Pruebas**: No debe existir ninguna característica en el programa que no haya sido probada, los programadores escriben pruebas para chequear el correcto funcionamiento del programa, los clientes realizan pruebas funcionales. El resultado, un programa más seguro que conforme pasa el tiempo es capaz de aceptar nuevos cambios.
- **Refactorización**: Permite a los equipos de programadores XP mejorar el diseño del sistema a través de todo el proceso de desarrollo. Los programadores evalúan continuamente el diseño y recodifican lo necesario. La finalidad es mantener un sistema enfocado a proveer el valor de negocio mediante la minimización del código duplicado y/o ineficiente.
- **Propiedad colectiva**: El código es de propiedad compartida. Nadie es el propietario de nada, todos son el propietario de todo. Este método difiere en mucho a los métodos tradicionales en los que un simple programador posee un conjunto de código. Los defensores de XP argumentan que mientras haya más gente trabajando en una pieza, menos errores aparecerán.

- **Integración continua**: Permite al equipo hacer un rápido progreso implementando las nuevas características del software. En lugar de crear versiones estables de acuerdo a un cronograma establecido, los equipos de programadores XP pueden reunir su código y reconstruir el sistema varias veces al día. Esto reduce los problemas de integración comunes en proyectos largos y estilo cascada.
- **40 horas semanales**: XP sostiene que los programadores cansados escriben código de menor calidad. Minimizar las horas extras y mantener los programadores frescos, generará código de mayor calidad.
- **Cliente en casa**: Al cliente se le dará poder para determinar los requerimientos, definir la funcionalidad, señalar las prioridades y responder las preguntas de los programadores. Esta fuerte interacción cara a cara con el programador disminuye el tiempo de comunicación y la cantidad de documentación, junto con los altos costes de su creación y mantenimiento. Este representante del cliente estará con el equipo de trabajo durante toda la realización del proyecto. (18)

XP no se basa en la planificación, el cliente forma parte del equipo de desarrollo del software, existen constantes cambios en los requisitos y es una metodología ágil apropiada para proyectos pequeños.

1.5.4 Metodología a utilizar

Para guiar el desarrollo de la solución se decide asumir la metodología XP basado en los aspectos vistos anteriormente y los que se relacionan a continuación:

- El cliente es parte del equipo de desarrollo.
- No existe un contrato tradicional al que darle cumplimiento.
- Es un proyecto considerado pequeño.

Con el propósito de definir cada una de las etapas y sus artefactos en el transcurso del desarrollo se decidió ajustar el ciclo de vida a la propuesta de solución como sigue:

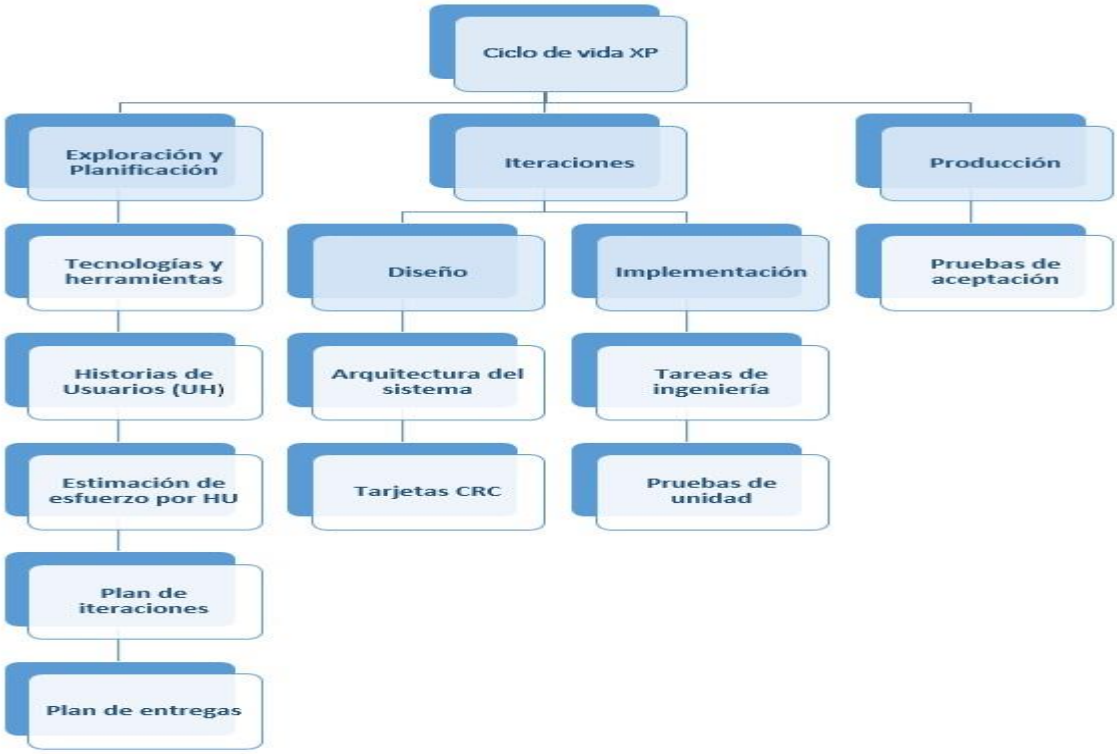


Figura 1: Ciclo de vida XP para guiar la propuesta de solución

En la Figura 1 se muestran las fases en que se dividirá el proceso de desarrollo del agente AASI, en la primera, Exploración y Planificación, se generarán los artefactos tecnologías y herramientas, historias de usuario (HU), estimación de esfuerzo por HU, plan de iteraciones y el plan de entregas; la segunda fase, Iteraciones, se divide en dos, Diseño e implementación, en Diseño se definirán la arquitectura del sistema y las tarjetas CRC y en implementación las tareas de la ingeniería y se realizarán las pruebas de unidad; y en la tercera y última fase, Producción, se realizarán las pruebas de aceptación.

1.6 Conclusiones del capítulo

La implementación de la herramienta AASI como solución para detectar las incidencias de seguridad informática en el Centro ISEC a tiempo, constituye la base de la propuesta de solución. Para ello se realizó un estudio de los conceptos asociados a las auditorías de seguridad informática, de las herramientas usadas internacionalmente para realizar auditorías llegando a la conclusión de que ninguna cumple en su totalidad con los objetivos propuestos y de las metodologías de desarrollo de software donde se define XP como la metodología de desarrollo a utilizar.

CAPÍTULO 2: EXPLORACIÓN Y PLANIFICACIÓN

La tendencia actual en el campo de la producción de software a nivel mundial es obtener productos de software en el menor tiempo posible con elevados estándares de calidad y elaborar la documentación estrictamente necesaria, por lo cual las metodologías de desarrollo ágiles surgen en aras de brindar a los desarrolladores una guía para alcanzar lo anteriormente expuesto.

En el presente capítulo, se aborda el entorno de desarrollo definido, se explican las diferentes características que va a presentar el sistema a implementar, sus funcionalidades siguiendo los aspectos que plantea la metodología ágil XP en sus fases de Exploración y Planificación, la cual servirá de guía para tener mayor organización en la distribución del tiempo, actividades y artefactos a desarrollar.

2.1 Tecnologías y herramientas

Con el objetivo de asegurar a lo largo de todo el proceso de desarrollo que el sistema cumpla con los requisitos tanto funcionales como no funcionales que fueron levantados conjuntamente con el cliente, se describen a continuación las tecnologías y herramientas que mejor se adecúan para el desarrollo del producto.

2.1.1 Lenguaje de programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes. (19)

Existen muchos y con disímiles funcionalidades y facilidades que favorecen el desarrollo de aplicaciones, entre los más usados a nivel internacional se encuentran c++, c#, y Java.

2.1.1.1 Java

Java es un lenguaje de programación de ordenadores, diseñado como una mejora de C++, y desarrollado por Sun Microsystems.

Se pretendía crear un lenguaje con algunas de las características básicas de C++, pero que necesitara menos recursos y que fuera menos propenso a errores de programación. Evolucionó hasta convertirse en un lenguaje muy aplicable a Internet y programación de sistemas distribuidos en general.

Pero su campo de aplicación no es exclusivamente internet, una de las grandes ventajas de Java es que se procura que sea totalmente independiente del hardware utilizando la Máquina Virtual de Java (Java Virtual Machine JVM por sus siglas en inglés) para su uso en distintos tipos de SO. Un programa en Java podrá funcionar en cualquier ordenador para el que exista la JVM, en ordenadores equipados con los SO Windows, Linux, Solaris y algún otro; incluso muchos teléfonos móviles actuales son capaces de usar programas creados en Java. A cambio, la existencia de ese paso intermedio hace que los programas Java no sean tan rápidos como puede ser un programa realizado en C, C++ o Pascal y optimizado para una cierta máquina en concreto.

Ventajas:

Independiente de la Plataforma

La principal característica de Java es que es independiente de la plataforma. Esto significa que cuando estás programando en Java, no necesitas conocer a priori el tipo de ordenador o el SO para el que estás programando. Puedes ejecutar el mismo programa en una PC con Windows, otro con Linux, en un Servidor SUN con SO Solaris, o en un teléfono móvil de última generación. Esta es una ventaja muy significativa, se logra mediante la llamada JVM.

Orientado a Objetos

Java es orientado a objetos. El paradigma de programación orientada a objetos supuso un gran avance en el desarrollo de aplicaciones, ya que es capaz de acercar la forma de programar a la forma de pensar del ser humano. Así, al igual que pensamos en objetos como una silla, una mesa o un coche, cuando programamos en un lenguaje orientado a objetos trabajamos con conceptos similares.

Gestión de Memoria

Existe un proceso independiente denominado “Garbage Collector” que se encarga de liberar automáticamente toda la memoria que ya no se utiliza, de manera que la liberación de memoria se hace de manera trasparente al programador.

Facilidad de Aprendizaje

El lenguaje Java es relativamente fácil de aprender, contando con la existencia de abundante documentación dentro de la escuela y en internet.

Entorno Integrado de Desarrollo (IDE)

Existen excelentes editores (IDEs) que aportan multitud de ayudas a la programación, haciendo que el desarrollo sea más fluido y cómodo. Ejemplos de excelentes IDEs son Eclipse, NetBeans, e IntelliJIdea.

Gestión de Errores

Una de las soluciones propuestas por el lenguaje Java a uno de los problemas recurrentes en otros lenguajes de programación es la gestión de errores a través de excepciones. Cuando se produce un evento excepcional, se altera el flujo normal de ejecución del programa y se gestiona un flujo alternativo de control de este tipo de errores.

Desventajas:

Se trata de un lenguaje con muchas cualidades, pero también tiene fisuras, así al tratarse de un lenguaje interpretado, el rendimiento en la ejecución de programas suele ser un poco menor.

Por los beneficios, las características que brinda y según las exigencias del cliente,.se decide usar Java como lenguaje de programación para implementar las funcionalidades de la aplicación AASI.

2.1.2 Entorno integrado de desarrolloa

Un IDE es un programa que comprende un entorno de programación amigable para uno o varios lenguajes, brindando facilidades al programador, tales como: un editor de código, un compilador, un depurador y, opcionalmente, un constructor de interfaz gráfica.

2.1.2.1 NetBeans

NetBeans es un entorno de desarrollo integrado con licencia GPL, modular y basado en estándares, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

Es un IDE sumamente completo, fácil de usar, cómodo y de excelente calidad, muy usado entre los programadores de Java hoy en día, por lo que hay mucha información al respecto.

Para la solución se utilizará en su versión 7.3.

2.1.3 Sistema gestor de base de datos

Un Sistema Gestor de Bases de Datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD

permite definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos.

2.1.3.1 SQLite

SQLite son librerías escritas en C, sin servidor, ni necesidad de configuración y el motor de base de datos transaccional de SQL. El código fuente para SQLite es de dominio público (20).

Ventajas:

- **Tamaño:** Tiene una pequeña memoria y una única biblioteca es necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas.
- **Rendimiento de base de datos:** Realiza operaciones de manera eficiente y es más rápido que MySQL y PostgreSQL.
- **Portabilidad:** Se ejecuta en muchas plataformas y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración.
- **Estabilidad:** Es compatible con ACID, reunión de los cuatro criterios de Atomicidad, Consistencia, Aislamiento y Durabilidad.
- **SQL (Standard Query Language):** (en español Lenguaje Estándar de Consultas). Implementa un gran subconjunto de la ANSI – 92 SQL estándar, incluyendo sub-consultas, generación de usuarios, vistas y triggers.
- **Costo:** Es de dominio público, y por tanto, es libre de utilizar para cualquier propósito sin costo y se puede redistribuir libremente.

Características:

- **No posee configuración:** De la forma en que fue creado y diseñado, no necesita ser instalado, prender, reiniciar o apagar un servidor, e incluso configurarlo. Esta cualidad permite que no haya un administrador de base de datos para crear las tablas, vistas, asignar permisos.
- **Portabilidad:** Puede ser ejecutado en diferentes sistemas operativos, como ser Windows, Linux, BSD, Mac OS X, Solaris, HPUX, AIX. Se pude notar que muchos de ellos trabajan a 16, 32 y 64 Bits. La portabilidad no está dada en sí por el software, sino por la base de datos condensada en un solo fichero, que puede estar situado en cualquier directorio, trayendo como ventaja que la

base de datos puede ser fácilmente copiada a algún dispositivo USB (Conductor Universal en Serie) o ser enviada vía correo electrónico.

- **Registros de longitud variable:** Generalmente los motores asignan una cantidad fija de espacio en disco para cada fila en la mayoría de los campos de una determinada tabla. En cambio, SQLite realizará todo lo contrario, utilizando para ello la cantidad de espacio en disco necesario para almacenar la información real del campo

Por los elementos planteados se decide usar SQLite en su versión 3.0 para almacenar localmente los datos generados después de cada auditoría.

2.1.4 Lenguaje de modelado

Para diagramar las entidades del sistema se usará Unified Modeling Language (UML, Lenguaje Unificado de Modelado), es un lenguaje utilizado para especificar, visualizar, construir y documentar artefactos de un sistema de software.

Visual Paradigm es una herramienta libre y fácil de usar, que emplea como lenguaje de modelado principal UML y facilita la interoperabilidad con otras aplicaciones.

Se utilizará esta herramienta en su versión 8.0 para confeccionar el modelo de datos por las facilidades que proporciona para la generación de artefactos.

2.1.5 Estándar de cifrado de datos (DES)

DES es un estándar de cifrado, algoritmo o método para encriptar datos e información. Se trata de un sistema de cifrado simétrico por bloques de 64 bits, de los que 8 bits (un byte) se utilizan como control de paridad para la verificación de la integridad de la clave. El algoritmo se encarga de realizar combinaciones, sustituciones y permutaciones entre el texto a cifrar y la clave, asegurándose al mismo tiempo de que las operaciones puedan realizarse en ambas direcciones para el descifrado. (21)

Características:

- Ofrece un alto nivel de seguridad relacionado con una pequeña clave utilizada para cifrado y descifrado.
- Comprensible.
- Adaptable.

- Económico y eficaz.

DES es un algoritmo que forma parte de la librerías nativas de Java por lo que se decide usar DES para el cifrado de los datos que son almacenados localmente después de cada auditoría.

2.1.6 Herramientas externas

Para la implementación de las funcionalidades es necesario el uso de herramientas externas que no se encuentran incluidas dentro del lenguaje de programación Java, a continuación se describen las herramientas que serán utilizadas para el desarrollo del agente AASI:

org.sqlite.JDBC:

Es un driver o manejador que permite la conexión con la BD SQLite.

com.enterprisedt.net.ftp:

Es una librería que permite la comunicación y escritura en el servidor FTP.

Junit:

Es un framework de código abierto para pruebas automáticas en Java, fácil de usar, sencillo y eficiente en las pruebas que realiza.

2.2 ASSI

A continuación se describen los actores del sistema según su rol y las acciones que podrán realizar, los requisitos funcionales que fueron identificados en trabajo conjunto con el cliente y las consultas al SO que son necesarias para obtener la información requerida.

2.2.1 Actores del sistema

Los actores del sistema difieren únicamente en el nivel de permisos que cada uno tiene. A continuación se muestra una descripción detallada de cada uno:

Personas relacionadas	Descripción
Usuario	Podrá realizar las auditorías manuales y ver los informes de auditoría que son guardados localmente, sin necesidad de permisos.

Administrador	Podrá realizar las auditorías manuales y ver los informes de auditoría que son guardados localmente. Además tendrá los permisos de configuración donde podrá seleccionar los tipos de políticas de seguridad que desea comprobar, la frecuencia con que se realizan las auditorías automáticas, exportar a un archivo con extensión txt la información de las auditorías realizadas y encriptarlo, y ver los log generados por los usuarios.

Tabla 1: Actores del sistema

2.2.2 Funcionalidades

Las funcionalidades del sistema abarcarán el flujo básico de la evaluación realizada por el AASI.

A continuación se detallan:

1. Realizar auditoría manual.
2. Realizar auditoría automática.
3. Configurar auditorías.
4. Comprobar que el nombre de la PC cumpla con la estructura detallada en la GCSI.
5. Comprobar que la PC esté unida al dominio.
6. Comprobar que las actualizaciones del SO cumplan con la estructura detallada en la GCSI.
7. Comprobar que la versión del SO cumpla con la estructura detallada en la GCSI.
8. Comprobar que el antivirus cumpla con la estructura detallada en la GCSI.
9. Comprobar la existencia en la PC de los Grupos descritos en la GCSI.
10. Comprobar que las cuentas de usuario cumplan con la estructura detallada en la GCSI.
11. Comprobar que el Sistema Básico de Entrada Salida (BIOS) cumpla con la estructura detallada en la GCSI.
12. Comprobar que el firewall cumpla con la estructura detallada en la GCSI.

13. Comprobar que el programa True Crypt cumpla con la estructura descrita en la GCSI.
14. Comprobar que el bloqueo de la PC cumpla con la estructura descrita en la GCSI.
15. Comprobar que los log de la PC cumplan con la estructura descrita en la GCSI.
16. Guardar los log de usuarios.
17. Mostrar alerta con el número de políticas de seguridad que no cumple la PC después de realizada una auditoría.

2.2.3 Consultas al Sistema Operativo (SO)

Para comprobar los elementos descritos en la GCSI se necesita conocer un conjunto de propiedades tanto de la PC como del SO, la forma de obtener esta información depende del SO en que se está ejecutando el agente, las consultas serán el medio por el cual se recogerá la información que se desea, este proceso se realizará a través de comandos y será totalmente invisible a los ojos del usuario.

Para los SO **Windows** se utilizarán generalmente las consultas WMI (Windows Management Instrumentation en español Instrumentación de la Administración de Windows) definidas por Microsoft, usadas para acceder al sistema, aplicaciones, redes y dispositivos de los SO Windows, específicamente se usarán las consultas WMIC, estas se ejecutan como comandos y heredan de la API (Interfaz de Programación de Aplicaciones) WMI. Además se usarán comandos nativos de Windows y en algunos casos se obtendrá la información a través del registro del sistema.

Para los SO **Unix** se ejecutarán comandos nativos ya que no es necesario el uso de otra herramienta para obtener la información requerida.

2.3 Historias de Usuario

Las historias de usuarios (HU) son la técnica utilizada en XP para describir los requisitos del software con pequeños textos en los que el cliente describe una actividad que realizará el sistema de forma sencilla y clara. Se puede considerar que las historias de usuario juegan un papel similar a lo que equivaldría a los casos de uso en el proceso unificado, pero son muy diferentes, pues muestran solamente la silueta de una tarea a realizarse.

Leyenda:

Número: Número de identificación para las HU, sería incremental en el tiempo.

Nombre Historia de Usuario: Es el nombre de la HU, sirve para identificarla fácilmente tanto para los desarrolladores como para los clientes.

Modificación de Historia de Usuario Número: Cantidad de modificaciones que se le ha realizado a la HU (de no tener modificaciones se pone ninguna, si no la cantidad de veces que ha sido modificada).

Usuario: Nombre del programador encargado de implementar la HU.

Prioridad del negocio: Qué tan importante es para el cliente, se clasifica en Muy alta, Alta y Media.

Riesgo de desarrollo: Qué tan difícil es para el desarrollador (Alto, Medio o Bajo).

Iteración asignada: Iteración a que corresponde, número de la iteración en la que se desarrollará la HU.

Puntos estimados: Tiempo en semanas que se le asignará. (Estimado)

Descripción: Es la descripción de la historia, detallando las operaciones del usuario y las respuestas del sistema.

Observaciones: Informaciones de interés, como glosarios, detalles del usuario, etc.

Prototipo de Interfaz: Contiene la imagen de una de las interfaces de usuario relacionadas con la HU.

A continuación se muestra la historia de usuario Realizar auditoría manual, las restantes se encuentran en el ***Anexo 1***.

Historia de Usuario	
Número: HU-1	**Nombre Historia de Usuario:** Realizar auditoría manual.
Modificación de Historia de Usuario Número: Ninguna.	
Usuario: Jeacny Reyes Echevarria.	**Iteración Asignada:** Primera.
Prioridad en el negocio: Muy alta.	**Puntos Estimados:** 1
Descripción: El agente debe mostrar a cualquiera de los actores del sistema, a través de una interfaz, los informes de las auditorías realizadas y la opción de realizar una auditoría manual; almacenando	

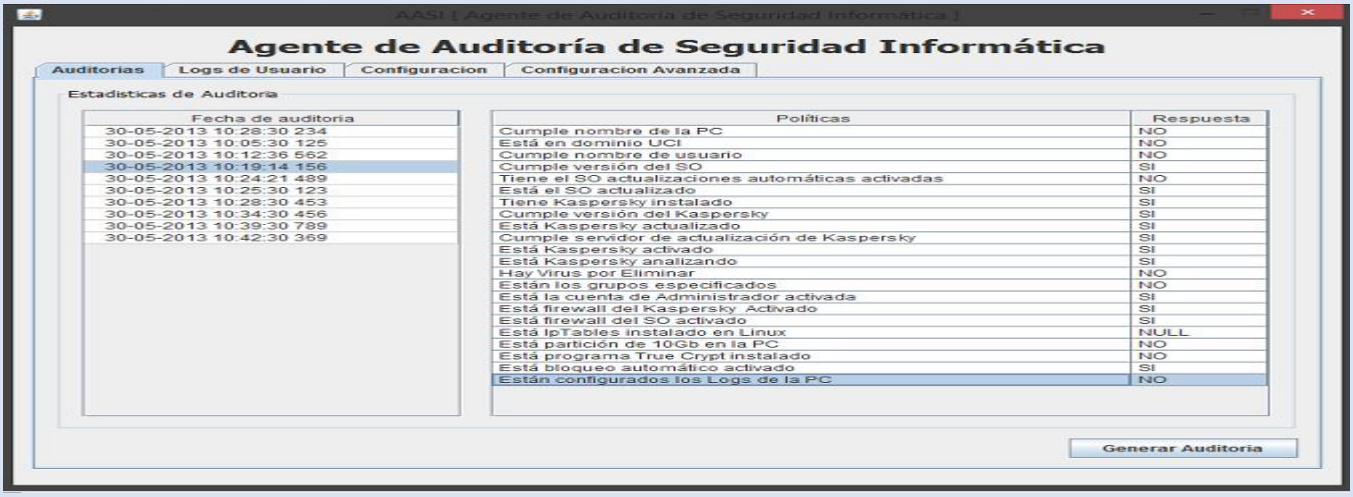
localmente los resultados en una base de datos con la fecha y hora en que se realizó la auditoría. Además enviará los resultados de la auditoría realizada a un servidor ftp en caso de existir conexión.
Observaciones.
Prototipo de Interfaz: 

Tabla 2.HU-1: Realizar auditoría manual.

2.4 Requisitos no funcionales

Los requisitos no funcionales son una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo.

Software:

La PC con SO Windows 7 donde se ejecute el agente debe tener habilitado el servicio WMI con sus respectivas dependencias.

La PC donde se ejecute el agente debe tener instalada la máquina virtual de Java (JVM) versión.

Restricciones del Sistema:

El agente debe iniciar con el SO y correr como un servicio.

Introducir la primera vez que se ejecute el programa, el Centro, proyecto, laboratorio y número de la PC donde se está ejecutando.

Portabilidad:

El agente podrá ser utilizado en los ordenadores que usen SO Windows y Linux que la Universidad brinda soporte.

Hardware:

La PC donde se ejecute el agente debe tener como mínimo 512 MB de RAM.

La PC donde se ejecute el agente debe tener como mínimo un CPU con un solo núcleo y 1.6Ghz de velocidad.

La PC donde se ejecute el agente debe tener un espacio mínimo de disco de 10 MB para almacenar el agente y de 5 MB para almacenamiento en la base de datos por un espacio de 5 años.

Apariencia e interfaz externa:

El sistema debe contar con una interfaz sencilla, fácil y cómoda que permita a los usuarios interactuar con el sistema, aún sin tener amplios conocimientos del funcionamiento de este.

Las interfaces evitarán cargar diálogos, paneles o ventanas innecesarias.

La fuente de la letra deberá ser uniforme para todas las interfaces.

Debe existir coherencia entre los íconos, los botones y la función que deben realizar.

Seguridad:

El agente encriptará la información que sea almacenada localmente.

El agente contará con permisos según los roles que desempeñen en el sistema, con el objetivo de solo permitir acceso a la configuración a los usuarios autorizados.

2.5 Estimación de esfuerzo por Historia de Usuario

La siguiente tabla contiene la estimación de esfuerzo por Historia de Usuario según el orden a realizar. Esta estimación incluye todo el esfuerzo asociado a la implementación de la HU, la misma expresa utilizando como medida el punto (máximo esfuerzo). Un punto se considera como una semana ideal de trabajo, donde se trabaje el tiempo planeado sin ningún tipo de interrupción.

Historia de Usuario	Puntos estimados

Realizar auditoría manual.	1
Realizar auditoría automática.	1
Configurar auditorías.	1
Comprobar nombre de la PC.	1
Comprobar dominio.	1
Comprobar actualizaciones.	1
Comprobar versión.	1
Comprobar antivirus.	1
Comprobar existencia de Grupos.	1
Comprobar cuentas de usuario.	1
Comprobar BIOS.	1
Comprobar firewall.	1
Comprobar True Crypt.	1
Comprobar bloqueo.	1
Comprobar log PC	1
Guardar log usuarios.	1
Mostrar alerta.	1

Tabla 3: Estimación de esfuerzo por Historia de Usuario

2.6 Plan de iteraciones

Luego de identificar las HU y la estimación del esfuerzo por cada una de ellas, se procede a realizar el plan de iteraciones, en el cual estarán contenidas las HU en el orden a realizar por cada iteración según su orden de relevancia, así como el total de semanas que durarán cada una de estas. Teniendo en cuenta el riesgo para desarrollar cada una de las historias de usuario, el tamaño del equipo de desarrollo, así como otros factores subjetivos, se decidió dividir el proyecto en tres iteraciones, detalladas a continuación:

Iteración 1: En esta iteración se implementarán las HU de prioridad muy alta, funcionalidades que son indispensables para cubrir las necesidades del cliente y que inciden críticamente en el funcionamiento de la aplicación.

Iteración 2: En esta iteración se implementarán las HU de prioridad alta, las cuales no son menos importantes que las mencionadas anteriormente pero debido al número de HU se determinó realizarlas en otra iteración.

Iteración 3: En esta iteración se implementarán las HU de prioridad media.

Iteraciones	Orden de las HU por iteración	Duración total de la iteración en semanas
Iteración 1	Realizar auditoría manual.	6
	Realizar auditoría automática.	
	Configurar auditorías.	
	Comprobar nombre de la PC.	
	Comprobar dominio.	
	Comprobar actualizaciones.	
	Comprobar versión.	
Iteración 2	Comprobar antivirus.	3
	Comprobar existencia de Grupos.	
	Comprobar cuentas de usuario.	
	Comprobar BIOS.	
	Comprobar firewall.	
	Comprobar True Crypt.	
	Comprobar bloqueo.	
	Comprobar log PC	
Iteración 3	Guardar log usuarios.	2
	Mostrar alerta.	

Tabla 4: Plan de Iteraciones

2.7 Plan de entregas

Este plan se obtiene a través de reuniones entre el equipo de trabajo y el cliente, para definir el marco temporal de la realización del sistema. Según la duración de cada iteración, así será la fecha aproximada de entrega.

A continuación se muestra una tabla con las fechas aproximadas de entregas por cada iteración.

Herramienta	Final 1ra Iteración 1ra semana de abril	Final 2da Iteración 4ta semana de abril	Final 3ra Iteración 4ta semana de mayo
AASI	0.1	0.4	1.0

Tabla 5: Plan de entregas

2.8 Conclusiones del capítulo

En este capítulo se describieron las fases de Exploración y Planeación para la propuesta de solución donde se definieron las tecnologías y herramientas que más se adecuaban a la solución del problema planteado en dependencia de sus características y necesidades: PostgreSQL 9.1 como sistema gestor de base de datos, Java como lenguaje de programación y NetBeans 7.3 como IDE de desarrollo. Además se obtuvieron los artefactos necesarios para la construcción de la herramienta: historias de usuario, plan de iteración, el esfuerzo estimado por HU y el plan de entregas.

CAPÍTULO 3: ITERACIONES Y PRODUCCIÓN

En el presente capítulo se procederá siguiendo la filosofía planteada por la metodología XP, según la cual la implementación debe realizarse de forma iterativa e incremental, obteniendo al final de cada iteración un producto funcional que debe ser examinado de conjunto con el cliente, pues de esta manera se garantiza la retroalimentación entre los desarrolladores y el cliente.

Se describe un modelo de despliegue para orientar a los usuarios durante el proceso de despliegue de la herramienta. Se puntualizan las iteraciones llevadas a cabo durante la etapa de construcción del sistema, definiéndose las tarjetas CRC como herramienta muy útil para propiciar el enfoque orientado a objetos y exponiéndose las tareas generadas por cada historia de usuario. Por último se realizarán las pruebas que contribuyen a la calidad de la aplicación en general.

3.1 Modelo de despliegue

A continuación se muestra el modelo de despliegue para el agente AASI. Este modelo muestra las relaciones físicas entre los componentes de hardware y software. Cuenta con una representación gráfica de los objetos físicos en tiempo de ejecución (nodos) y los dispositivos que lo componen, así como las relaciones entre ellos.

Figura 2 Diagrama de despliegue.

Nodo	Descripción
	Computadora donde se ejecutará el agente AASI y realizará las auditorías. **Sistema Operativo:** ✓ Linux - Estaciones de trabajo del Centro con: Nova, Ubuntu, Debian, Fedora, Kubuntu, Lubuntu, CentOS. - Servidores del Centro con: Nova Server y Ubuntu Server sin interfaz gráfica.

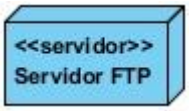
	✓ Windows: - Estaciones de trabajo del Centro con: Windows XP, Windows 7 y Windows 8. **Requisitos mínimos de hardware:** ✓ 512 MB de memoria RAM. ✓ Procesador 1.6 MHz ✓ 15 MB de almacenamiento. **Condiciones de ejecución:** ✓ Instalada la JVM versión 7. ✓ Para Windows 7: - Habilitado el servicio WMI y sus respectivas dependencias. **Pasos a seguir:** ✓ Ejecutar en la carpeta Agente el archivo AASI_start.bat en Windows y AASI_start.sh en Linux. ✓ Una vez iniciado el agente introducir en la sección Configuración el usuario: admin y la contraseña: admin para acceder a la sección Configuración Avanzada donde debe especificar el Centro, proyecto, laboratorio y # de la PC donde se está ejecutando el agente. Se recomienda cambiar usuario y contraseña para permitir acceso, sólo al personal autorizado.
	Computadora Servidor donde se almacenarán los resultados de las auditorías realizadas a todas las PC donde se encuentre ejecutándose el agente AASI. **Requisitos mínimos de hardware:** ✓ 512 MB de memoria RAM. ✓ Procesador 1.6 MHz ✓ 1 GB de almacenamiento por espacio de 5 años.

Tabla 6. Descripción de los nodos.

3.2 Arquitectura de software

La arquitectura de software es la organización fundamental de un sistema encarnado en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (22).

Toda arquitectura de software debe describir diversos aspectos del software, y generalmente, cada uno de estos se describe de una manera más comprensible si se utilizan distintos modelos o vistas.

3.2.1 Estilos arquitectónicos

Se hizo un estudio de los estilos arquitectónicos para decidir cuál usar en el desarrollo de la aplicación. A continuación se describen los más significativos para el entendimiento del concepto y la toma de la decisión final.

3.5.1.1 Arquitectura Monolítica:

Esta arquitectura se desarrolla cuando el software se estructura en grupos funcionales muy acoplados. No hay distribución, tanto a nivel físico como a nivel lógico. Está formado por la presentación, los datos y el procesamiento, una arquitectura rígida de programación en un solo componente y un único computador (23).

Es la arquitectura de los primeros sistemas operativos constituidos fundamentalmente por un solo programa compuesto de un conjunto de rutinas entrelazadas de tal forma que cada una puede llamar a cualquier otra. Las características fundamentales de este tipo de arquitectura son (23):

- Construcción del programa final a base de módulos compilados separadamente que se unen a través del ligador.
- Buena definición de parámetros de enlace entre las distintas rutinas existentes, que puede provocar mucho acoplamiento.
- Generalmente están hechos a medida, por lo que son eficientes y rápidos en su ejecución y gestión, pero por lo mismo carecen de flexibilidad para soportar diferentes ambientes de trabajo o tipos de aplicaciones.
- Difícil de depurar, ya un error en una función se puede manifestar en otra distinta.

Es una arquitectura donde todas las clases se encuentran acopladas en un solo componente, no permite la reutilización del código, provoca mucho acoplamiento y un cambio en una clase repercute en las demás haciendo difícil su depuración en caso de error, por lo que se desecha como arquitectura para describir los componentes que implementan las funcionalidades del agente AASI.

3.5.1.2 Cliente-Servidor:

La arquitectura Cliente/Servidor es el procesamiento cooperativo de la información por medio de un conjunto de procesadores, en el cual múltiples clientes, distribuidos geográficamente, solicitan requerimientos a uno o más servidores centrales. (23)

Se puede definir como una arquitectura distribuida en la que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes, permitiendo a los usuarios finales obtener acceso a la información de forma transparente. Ofrece ventajas de tipo organizativo debido a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

Un cliente es todo proceso que reclama servicios de otro, es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor.

Un servidor es todo proceso que proporciona un servicio a otros, es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles o capas y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Los niveles o capas pueden referirse a la distribución física o lógica de los componentes de un software, la descripción anterior y la que a continuación se hace se refiere a la distribución lógica de dichos componentes.

Existe varios tipos de arquitecturas para el estilo cliente-servidor como son:

Arquitectura en 2 niveles:

La arquitectura en 2 niveles se utiliza para describir los sistemas cliente/servidor en donde el cliente solicita recursos y el servidor responde directamente a la solicitud, con sus propios recursos, esto significa que el servidor no requiere otra aplicación para proporcionar parte del servicio. La capa de presentación interactúa con la capa de lógica de negocio; la misma presenta una interfaz para brindar servicios a la capa de presentación.

Arquitectura en 3 niveles:

La arquitectura en 3 niveles es una variación de la arquitectura cliente servidor donde, existe un nivel intermedio. Contiene una capa de Presentación para la comunicación con el usuario, la cual solicita y obtiene información de la capa inferior llamada Negocio donde se establecen las reglas que deben cumplirse, le brinda a la presentación la información que solicita y se comunica con la capa de Datos recogiendo los datos necesarios para cumplir con la solicitud del usuario; la capa de Datos contiene la información persistente y la comunicación con los componentes que contienen dicha información. Entre las ventajas que ofrece este estilo, donde cada capa cumple con un rol y tiene responsabilidades bien definidas, se encuentran: en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado y en caso de errores detectarlos con relativa facilidad.

3.2.2 Arquitectura del sistema

Para el sistema se adoptó una arquitectura multiniveles, específicamente de 3 niveles o capas, dichas capas son: la Capa de Presentación, que contiene las vistas de la aplicación y es la encargada de la comunicación con el usuario; la capa de Lógica de Negocio, la cual alberga las clases donde se implementan las funcionalidades del sistema y es la responsable de proporcionar la comunicación entre las capas de Presentación y la de Acceso a Datos, que contiene las clases encargadas de gestionar el acceso a los datos proporcionado a la capa Lógica de Negocio la información que solicite. Adicionalmente se define una capa transversal al resto de las capas, que contiene las clases relacionadas con la seguridad del sistema y las clases del dominio.

Figura 3. Arquitectura en capas/3 capas

Capa de presentación: Esta capa presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso, realiza un filtrado previo para comprobar que no hay errores de formato. Esta capa se comunica únicamente con la capa de negocio, y debe tener la característica de ser entendible y fácil de usar para el usuario.

Capa de Lógica negocio: La capa de lógica negocio está formada por interfaces y clases que definen cada uno de los componentes de negocio. Con el objetivo de mantener al mínimo posible el acoplamiento entre una capa y otra de la arquitectura se define una interfaz en la cual se establece el contrato del componente de negocio, dándole luego una implementación concreta. De esta forma, todas las dependencias entre capas se establecen a nivel de interfaces y no de implementaciones concretas.

Capa de datos: La capa de acceso a datos es la encargada de encapsular todos y cada uno de los componentes de acceso a datos, y es la encargada de acceder a los mismos.

Transversal: La capa transversal contiene las clases asociadas con la seguridad del sistema las cuales implementan los métodos para cifrar los datos almacenados localmente, además se implementan las clases del dominio que contienen los datos persistentes de la aplicación.

La clave del desarrollo en capas es que una capa solamente debe utilizar lo que la interfaz de la o las capas inferiores le brindan, de este modo se pueden intercambiar las capas respetando la interfaz.

3.2.3 Patrones de diseño

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

3.1.3.1 Patrones Grasp

“Son patrones generales de software para asignación de responsabilidades, es el acrónimo de "General Responsibility Assignment Software Patterns". Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software “. (24)

- **Patrón Controlador**: Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.
- **Alta cohesión:** Se refiere a que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la ella misma.
- **Bajo acoplamiento:** Se refiera a tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. Este patrón es utilizado en la relación entre capas ya que cada una de ellas interactúa solo con la inmediata inferior y mediante interfaces pudiendo así realizar cambios en una capa sin que la otra se vea afectada.
- **Experto**: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. De esta manera se garantiza una alta cohesión y un bajo acoplamiento. Este patrón es utilizado en casi todo el sistema debido a que la información necesaria para el cumplimiento de las funcionalidades está distribuida por todo el sistema. Un ejemplo de donde se observa la aplicación de este patrón es en las clases de la capa de acceso a datos ya que estas son las únicas encargadas de interactuar con la base de datos para efectuar diferentes acciones.

3.1.3.2 Patrones GOF (Gang of Four)

- **Fachada:** Se utiliza para proporcionar una interfaz unificada para un conjunto de funcionalidades, facilitando así su uso. Además simplifica el acceso al conjunto de funcionalidades ya que la comunicación se realiza a través de una única interfaz.

3.3 Modelo de datos

El modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Este describe las estructuras de datos de la base de datos correspondiente al contenido del sistema, permite describir los elementos que intervienen y la forma en que se relacionan estos elementos entre sí. A continuación se muestra el modelo de datos de la aplicación.

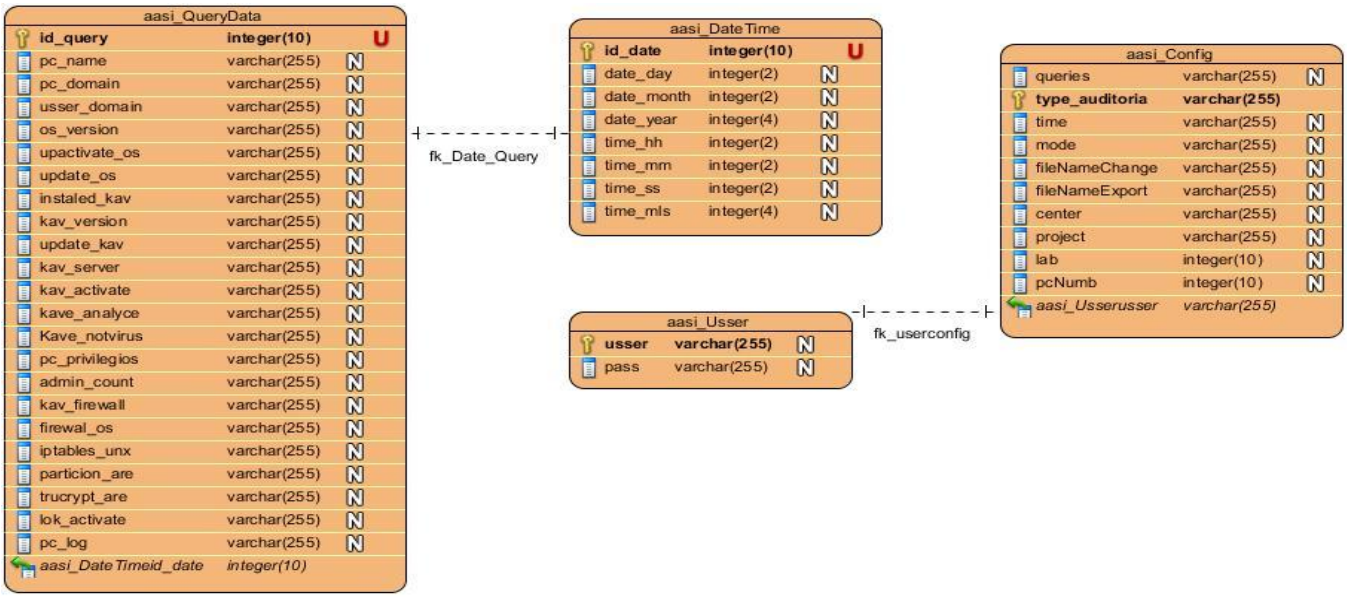


Figura 4. Modelo de datos

3.4 Tarjetas CRC (Clase, Responsabilidad, Colaborador)

La principal tarea de las tarjetas CRC es propiciar el enfoque orientado a objetos y dejar el pensamiento procedimental. Las tarjetas están divididas en tres secciones: nombre de la clase, responsabilidades y colaboradores.

Una clase describe cualquier objeto o evento, mediante los atributos y los métodos, las responsabilidades son las tareas que realizan o los métodos correspondientes a la clase y los colaboradores son las demás clase con las que trabaja conjuntamente para cumplir con sus responsabilidades.

FTPManager	
Responsabilidad	**Colaborador**
Almacenar los resultados de las auditorías realizadas en un servidor ftp.	com.enterprisedt.net.ftp

Tabla 7. Tarjeta CRC: FTPManager

SqlManager	
Responsabilidad	**Colaborador**
Ejecutar las acciones sobre la base de datos SQLite.	ninguno

Tabla 8: Tarjeta CRC: SqlManager

Conexion	
Responsabilidad	**Colaborador**
Conectar con la base de datos SQLite.	org.sqlite.JDBC

Tabla 9: Tarjeta CRC: Conexión

FileManager	
Responsabilidad	**Colaborador**
Gestionar las acciones con los archivos.	ninguno

Tabla 10: Tarjeta CRC: FileManager

Encoding	
Responsabilidad	**Colaborador**

Codificar y decodificar tanto una palabra como un texto completo.	ninguno

Tabla 11: Tarjeta CRC: Encoding

ASSIManager	
Responsabilidad	**Colaborador**
Gestionar todos los métodos para hacer cumplir las funcionalidades del agente.	ISqlManager IFileManager IDomain IEncoding

Tabla 12: Tarjeta CRC: ASSIManager

ValidateData	
Responsabilidad	**Colaborador**
Comprobar que la información del SO cumpla con la GCSI.	ninguno

Tabla 13: Tarjeta CRC: ValidateData

WinQuery	
Responsabilidad	**Colaborador**
Recoger la información del SO windows, la valida y la devuelve en una entidad.	ValidateData QueryData

Tabla 14: Tarjeta CRC: WinQuery

UnixQuery	
Responsabilidad	**Colaborador**
Recoger la información del SO Linux, la valida y la devuelve en una entidad.	ValidateData QueryData

Tabla 15: Tarjeta CRC: UnixQuery

ConfigManager	
Responsabilidad	**Colaborador**
Cargar y guardar la configuración del agente.	Configuration

Tabla 16: Tarjeta CRC: ConfigManager

UsserLogs	
Responsabilidad	**Colaborador**
Generar los logs a partir de las trazas seguidas por los usuarios.	Todas las clases del sistema menos las entidades.

Tabla 17: Tarjeta CRC: LogerManager

CommandLine	
Responsabilidad	**Colaborador**
Ejecutar las líneas de comando para los SO Unix.	ninguno

Tabla 18: Tarjeta CRC: CommandLine

PasrseProperties	
Responsabilidad	**Colaborador**
Seleccionar de la información recogida después de ejecutar un comando, la propiedad que se desea comprobar.	IFileManager

Tabla 19: Tarjeta CRC: PasrseProperties

MainForm	
Responsabilidad	**Colaborador**

Manejar los controles de la interfaz de usuario.	IAASIManager

Tabla 20: Tarjeta CRC: MainForm

AASI	
Responsabilidad	**Colaborador**
Mostrar la Interfaz de usuario y garantizar que la aplicación se ejecute como un servicio.	MainForm

Tabla 21: Tarjeta CRC: AASI

AASIConsole	
Responsabilidad	**Colaborador**
Realizar auditorías en las computadoras sin interfaz gráfica, a través de la consola.	IAASIManager

Tabla 22. Tarjeta CRC: AASIConsole

3.5 Tareas de ingeniería

Para definir las tareas de ingeniería se cuenta con una plantilla, la cual permite definir cada una de las actividades que estarán asociadas a las historias de usuario y que permitirán su implementación. A continuación se muestran las tareas asignadas por iteraciones.

3.5.1 Iteración 1

Durante esta iteración se abordaron las HU de máxima prioridad donde se construyó la base de la arquitectura del producto con las funcionalidades primarias necesarias para ser mostrado al cliente y obtener una rápida y amplia retroalimentación. A continuación se describen en la siguiente tabla:

Historia de Usuario	Estimación	Real
Realizar auditoría manual.	1	1
Realizar auditoría automática.	1	1

Configurar auditorías.	1	1
Comprobar nombre de la PC.	1	1
Comprobar dominio.	1	1
Comprobar actualizaciones.	1	1
Comprobar versión.	**1**	**1**

Tabla 23: HU se implementarán en la iteración 1

Historia de Usuario	Tareas de Ingeniería
Realizar auditoría manual.	Diseño interfaz principal. Implementar auditoría manual.
Realizar auditoría automática.	Implementar auditoría automática.
Configurar auditorías.	Autenticar usuario. Diseño interfaz de configuración.
Comprobar nombre de la PC.	Implementar Nombre PC.
Comprobar dominio.	Implementar Dominio.
Comprobar actualizaciones.	Implementar Actualizaciones.
Comprobar versión.	Implementar Versión.

Tabla 24: Tareas de Ingeniería para la Iteración 1

Las tareas a desarrollar por cada una de las historias de usuario anteriores son:

A continuación se detallan las tareas de ingeniería relacionadas con la HU- Realizar auditoría manual para la *Iteración1*, el resto de las tareas de esta iteración se muestran en el ***Anexo 2***.

Tareas de la Ingeniería	
Número de la tarea: 1	**Número de Historia de Usuario:** 1
Nombre de la tarea: Diseño de interfaz principal.	
Tipo de tarea: Desarrollo	**Puntos Estimados**: 0.5.
Fecha inicio: 4/3/2013.	**Fecha fin**: 9/3/2013
Programador responsable: Jeacny Reyes Echevarria.	

Descripción: Se implementarán las funcionalidades que permitan mostrar una interfaz donde a las personas relacionadas con el sistema se les brinde la opción de realizar las auditorías manuales, ver los informes de auditoría que son guardados localmente, y la opción de configuración donde solo se podrá acceder con la contraseña del usuario administrador.

Tabla 25: HU-1.Tarea de ingeniería No.1: Implementar auditoría manual

Tareas de la Ingeniería	
Número de la tarea: 1	**Número de Historia de Usuario:** 1
Nombre de la tarea: Implementar auditoría manual.	
Tipo de tarea: Desarrollo	**Puntos Estimados**: 0.5
Fecha inicio: 11/3/2013.	**Fecha fin**: 16/3/2013
Programador responsable: Jeacny Reyes Echevarria.	
Descripción: Se implementarán las funcionalidades que permitan comprobar en el instante las políticas de seguridad que se encuentran seleccionadas en la interfaz de configuración, almacenando localmente los resultados en una base de datos con la fecha y hora en que se realizó la auditoría y enviando los resultados de la auditoría realizada a un servidor ftp en caso de existir conexión.	

Tabla 26: HU-2.Tarea de ingeniería No.1: Implementar auditoría manual

3.5.2 Iteración 2

Durante esta iteración se abordaron las HU de mayor prioridad donde se definen las tareas de desarrollo de las mismas. Al culminar se consta de un producto casi listo para su puesta en funcionamiento con la mayoría de sus funcionalidades más críticas ya implementadas. A continuación se describe en la siguiente tabla:

Historia de Usuario	Estimación	Real
Comprobar antivirus.	1	1
Comprobar existencia de Grupos.	1	1
Comprobar cuentas de usuario.	1	1
Comprobar BIOS.	1	1
Comprobar firewall.	1	1

Comprobar True Crypt.	1	1
Comprobar bloqueo.	1	1
Comprobar log PC.	1	1

Tabla 27: HU- se implementarán en la Iteración 2

Las tareas a desarrollar por cada una de las historias de usuario anteriores son:

Historias de Usuario	Tareas de Ingeniería
Comprobar antivirus.	Implementar antivirus.
Comprobar existencia de Grupos.	Implementar existencia de Grupos.
Comprobar cuentas de usuario.	Implementar cuentas de usuario.
Comprobar BIOS.	Implementar BIOS.
Comprobar firewall.	Implementar firewall.
Comprobar True Crypt.	Implementar True Crypt.
Comprobar bloqueo.	Implementar bloqueo.
Comprobar log PC.	Implementar log PC

Tabla 28: Tareas de Ingeniería para la Iteración 2

A continuación se detallan las tareas de ingeniería relacionadas con la HU- Comprobar antivirus para la *Iteración 2*, el resto de las tareas de esta iteración se muestran en el ***Anexo 2.***

Tareas de la Ingeniería	
Número de la tarea: 1	**Número de Historia de Usuario:** 8
Nombre de la tarea: Implementar Antivirus.	
Tipo de tarea: Desarrollo	**Puntos Estimados**: 1.
Fecha inicio: 10/4/2013.	**Fecha fin**: 11/4/2013.
Programador responsable: Jeacny Reyes Echevarria.	
Descripción: Se implementarán las funcionalidades que permitan comprobar que el antivirus cumpla con la estructura detallada en la GCSI.	

Tabla 29: HU-8.Tarea de ingeniería No.1: Implementar Antivirus

3.5.3 Iteración 3

En el transcurso de esta iteración se implementan las HU referente a la tercera iteración que involucra facilidades que brinda la aplicación. Al culminar esta, se consta de un producto listo para su puesta en funcionamiento.

Historia de Usuario	Estimación	Real
Guardar log usuarios.	1	1
Mostrar alerta.	1	1

Tabla 30: HU se implementarán en la Iteración 3

Las tareas a desarrollar por cada una de las historias de usuario anteriores son:

Historias de Usuario	Tareas de Ingeniería
Guardar log usuarios.	Implementar log usuarios.
Mostrar alerta.	Implementar Alerta.

Tabla 31: Tareas de Ingeniería para la Iteración 3

A continuación se detallan las tareas de ingeniería relacionadas con las HU para la *Iteración 3*.

Tareas de la Ingeniería	
Número de la tarea: 1	**Número de Historia de Usuario:** 16
Nombre de la tarea: Implementar log usuarios.	
Tipo de tarea: Desarrollo	**Puntos Estimados**: 1
Fecha inicio: 1/5/2013.	**Fecha fin**: 11/5/2013
Programador responsable: Jeacny Reyes Echevarria.	
Descripción: Se implementarán las funcionalidades que permitan que los log generados por los usuarios, teniendo en cuenta los eventos relevantes, se almacenen localmente, encriptados, en un fichero con extensión txt. Además con permisos de administración se mostrará en la interfaz principal los log que están almacenados	

Tabla 32: HU-16.Tarea de ingeniería No.1: Implementar Log usuarios

Tareas de la Ingeniería	
Número de la tarea: 1	**Número de Historia de Usuario:** 17
Nombre de la tarea: Implementar Alerta.	
Tipo de tarea: Desarrollo	**Puntos Estimados**: 1
Fecha inicio: 13/5/2013.	**Fecha fin**: 18/5/2013
Programador responsable: Jeacny Reyes Echevarria.	
Descripción: Se implementarán las funcionalidades que permitan mostrar una alerta con los problemas detectados después de cada auditoría realizada.	

Tabla 33: HU-17.Tarea de ingeniería No.1: Implementar Alerta

3.6 Pruebas

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas. Los resultados son observados, registrados, y una evaluación es hecha de algún aspecto del sistema o componente (25).

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. Está enfocada principalmente en la evaluación y determinación de la calidad del producto. Estas están definidas en estrategias, niveles, tipos, métodos y técnicas de prueba (22).

3.6.1 Estrategia de Prueba

La realización de pruebas en XP anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones. XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y

diseñada por los programadores y pruebas de aceptación o pruebas funcionales, destinadas a evaluar si al final de una iteración se logró implementar las funcionalidades requeridas por el cliente final (26).

La estrategia de prueba describe el enfoque y los objetivos generales de las actividades de prueba. Incluye los niveles de prueba (unidad, integración, etc.) a ser diseccionados y el tipo de prueba a ser ejecutadas (funcional, stress, etc.).

La estrategia de prueba define:

- Técnicas de pruebas (manual o automática) y herramientas a ser usadas.
- Qué criterios de éxitos y culminación de la prueba serán usados.
- Consideraciones especiales afectadas por requerimientos de recursos o que tengan implicaciones en la planificación.

La estrategia a seguir para la realización de las pruebas al sistema contempla dos niveles de pruebas: el nivel de pruebas de Unidad y el nivel de pruebas de Aceptación.

Pruebas unitarias:

Enfocada a los elementos probables más pequeños del software. Aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca. Todo el código debe tener pruebas unitarias y debe pasarlas antes de ser lanzado. Las pruebas unitarias se realizarán a las HU desarrolladas después de culminada cada iteración y se utilizará para ello la herramienta Junit.

Pruebas de aceptación:

Son realizadas por el usuario final con el objetivo de evaluar el producto, son pruebas de caja negra en donde el cliente verifica que lo que se está probando funcione correctamente. Verifican el comportamiento completo del sistema o de la aplicación en cuestión. Cuando se pasa la prueba de aceptación se considera la historia de usuario finalizada. Las pruebas de aceptación se realizarán al final de la fase de iteraciones, con el software funcionando como un todo y el cliente como probador.

3.7 Iteraciones

XP propone desarrollar las HU en iteraciones para permitir la evolución del sistema y capturar los defectos a tiempo. Se decidió dividir el proceso de desarrollo del agente AASI en tres iteraciones, realizándose al

final de cada iteración pruebas al código de las HU implementadas para verificar su correcto funcionamiento, usando la herramienta Junit para realizar las pruebas automáticas.

3.7.1 Primera iteración

En la primera iteración se implementaron las HU siguientes: Realizar auditoría manual, Realizar auditoría automática, Configurar auditorías, Comprobar nombre de la PC, Comprobar dominio, Comprobar actualizaciones, y Comprobar versión.

Se usaron las consultas WMIC para obtener la información requerida de los sistemas operativos y comprobar que cumplan con las políticas descritas en la GCSI, y el driver org.sqlite.JDBC para conectarse con la BD y almacenar los resultados de las auditorías realizadas.

Las HU Realizar auditoría manual y Realizar auditoría automática no se desarrollaron completas puesto que dependen de los resultados de las comprobaciones de las HU de la segunda iteración, además de las restantes HU de la primera iteración.

3.7.1.1 Pruebas unitarias primera iteración

Se realizaron las pruebas al código de la primera iteración, las HU Realizar auditoría manual y automática no se pudieron probar de forma automática, usando la herramienta Junit, ya que estas usan clases que no se pudieron pasar por parámetro en los métodos que las implementan, pero si se comprobó su funcionamiento a partir de pruebas manuales.

Los problemas detectados en esta iteración fueron:

- No se obtuvieron los resultados requeridos del SO usando la misma consulta WMIC en diferentes versiones de Windows.
- Las auditorías tanto manual como automática no insertan correctamente los datos en la BD.
- El resultado de comprobar el nombre de la PC y la versión del sistema operativo no se corresponde con el real.
- No se permite ejecutar desde la capa de presentación métodos de la capa de lógica de negocio que a su vez utilizan métodos de la capa de acceso a datos y la capa de seguridad.

A estas deficiencias se les dio solución, arrojando finalmente resultados alentadores en iteraciones de prueba posteriores. A continuación se muestra el resultado de las pruebas automáticas realizadas a las demás HU de esta iteración.

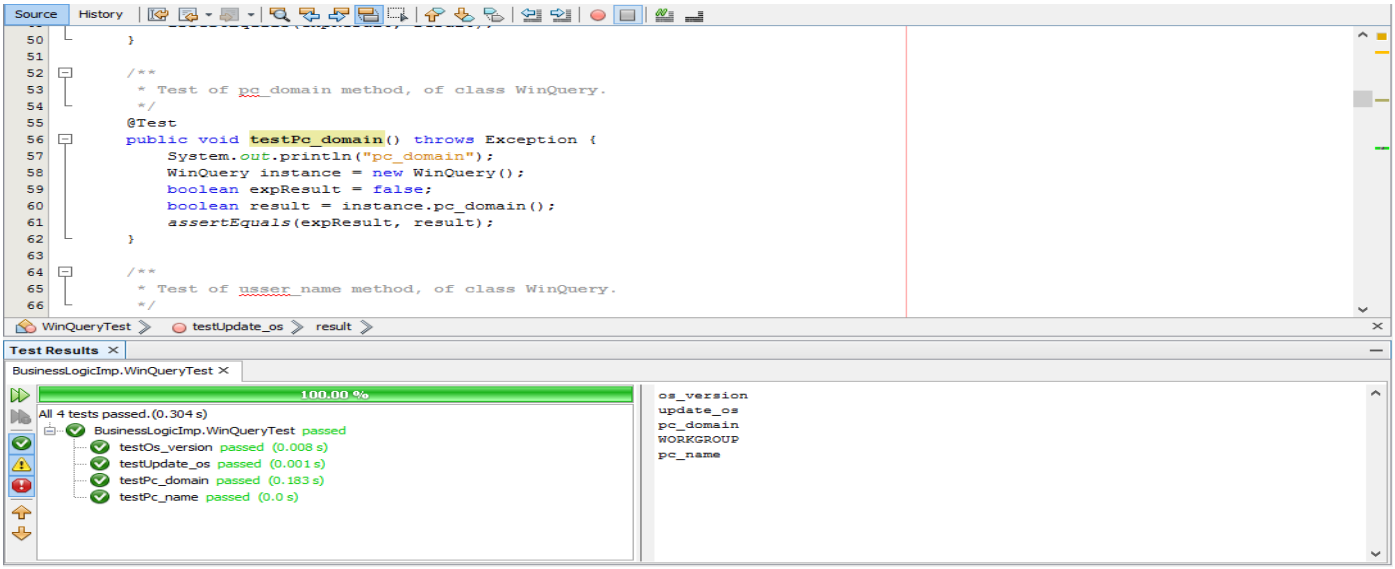


Figura 5. Pruebas automáticas 1ra iteración

3.7.2 Segunda iteración

En la segunda iteración se implementaron las HU siguientes: Comprobar antivirus, Comprobar existencia de Grupos, Comprobar cuentas de usuario, Comprobar BIOS, Comprobar firewall, Comprobar True Crypt, Comprobar bloqueo, Comprobar log PC.

Se usaron las consultas WMIC para obtener la información requerida de los SO y comprobar que cumplan con las políticas descritas en la GCSI, además de consultas específicas utilizando el registro de windows para comprobar las políticas que debe cumplir el antivirus Kaspersky. También se terminó de implementar las HU Realizar auditoría manual y Realizar auditoría automática.

3.7.2.1 Pruebas unitarias segunda iteración

Se realizaron las pruebas al código de las HU de la segunda iteración detectándose las siguientes deficiencias:

- El resultado de leer el fichero para obtener la propiedad en las consultas ejecutadas al SO no coincide con el resultado esperado.

- El resultado de comprobar algunas de las políticas que debe cumplir Kaspersky para los sistemas operativos Windows no corresponde con el real.

Después de haber solucionado las no conformidades se realizaron varias iteraciones de pruebas automáticas arrojando resultados positivos. A continuación se muestran dichas pruebas.

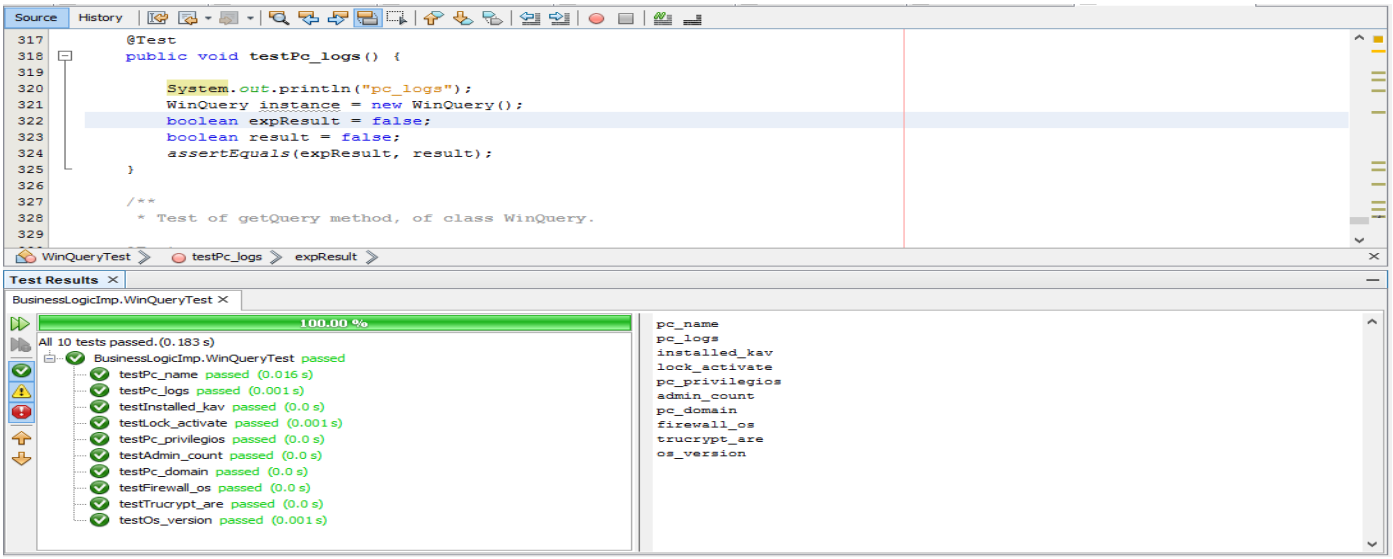


Figura 6. Pruebas automáticas 2da iteración

3.7.3 Tercera iteración

En la tercera iteración se implementaron las HU Guardar log usuarios y Mostrar alerta, para ello se implementó la clase UsserLog usando las librerías nativas de Java, para gestionar las trazas generadas por los usuarios.

3.7.3.1 Pruebas unitarias tercera iteración

Se realizaron las pruebas unitarias a las HU de la tercera iteración detectándose el siguiente problema:

- El resultado que muestra el método ReadLog usado para leer los log generados por los usuarios, no coincide con el real.

A este problema se le dio solución y en las siguientes iteraciones de prueba se obtuvieron los resultados esperados. A continuación se muestra el resultado de las pruebas al código de esta iteración.

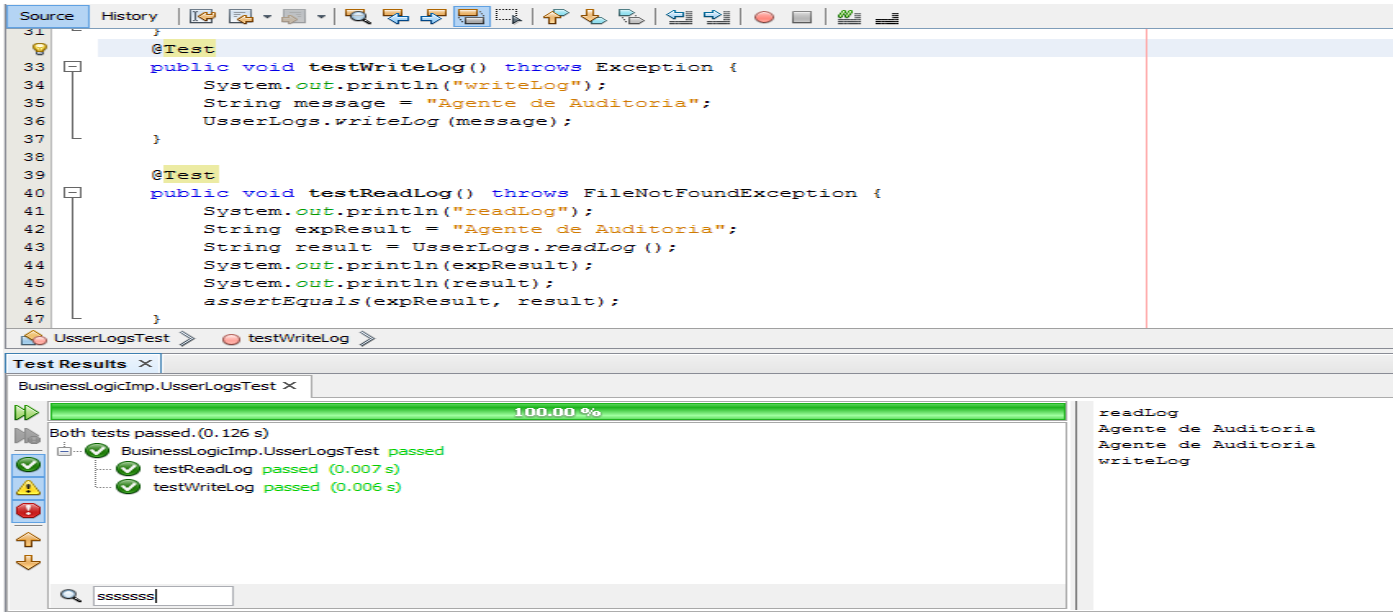


Figura 7. Pruebas automáticas 3ra iteración

3.8 Pruebas de aceptación

Las pruebas de aceptación se realizaron con el software funcionando como un todo, dándole un uso real a la aplicación y el cliente como probador durante todo el proceso, los errores detectados fueron reportados como defectos.

Se realizaron con el objetivo de mostrar al cliente que el producto cumple con los requerimientos funcionales. Para realizar dichas pruebas se usaron los casos de prueba de aceptación donde se describe el proceso de prueba a realizar, las deficiencias identificadas en las primeras iteraciones de pruebas de aceptación fueron:

- La forma en que se muestran las auditorías realizadas no es agradable a los ojos de los usuarios.
- No se le informa al usuario cuando se intenta guardar las configuraciones sin haber seleccionado ninguna política a comprobar.
- No se le informa al usuario cuando se ha realizado una auditoría.
- No se permite mostrar en la interfaz principal los log generados por los usuarios.
- No se deniega el acceso a las configuraciones cuando se selecciona el botón Salir.

Los defectos encontrados por el cliente fueron solucionados y en las siguientes iteraciones de prueba, también con el cliente como probador, se obtuvieron resultados satisfactorios. Los resultados de la última iteración de prueba de aceptación se muestran en el **Anexo 3**.

3.9 Conclusiones del capítulo

En este capítulo se definió la arquitectura de tres niveles para la construcción del agente, se confeccionaron las tarjetas CRC y las tareas de la ingeniería las cuales permiten definir cada una de las actividades que estarán asociadas a las historias de usuario y que permitieron la implementación de estas. Además se realizaron las pruebas al código de la aplicación usando la herramienta Junit y las pruebas de aceptación conjuntamente con el cliente, las mismas arrojaron en las primeras iteraciones de prueba, siete no conformidades en las pruebas unitarias y cinco defectos en las pruebas de aceptación.
A las no conformidades detectadas se les dio solución y en posteriores iteraciones de prueba se obtuvieron finalmente resultados positivos.

CONCLUSIONES GENERALES

En el presente trabajo fueron cumplidos los objetivos planteados, así como las tareas de la investigación, arribando a las siguientes conclusiones:

- Se realizó una revisión bibliográfica y análisis de los fundamentos teóricos referentes a seguridad informática, auditorías informáticas y auditorías de seguridad informáticas.
- Se hizo una evaluación de las herramientas existentes para la realización de auditorías de seguridad informáticas, determinándose que ninguna solucionaba el problema detectado en el Centro ISEC.
- Se seleccionó la metodología, herramientas y tecnologías que mejor se ajustaban para la creación de un Agente de Auditorías de Seguridad Informática.
- Se seleccionó la arquitectura de software a emplear para la construcción de un Agente de Auditorías de Seguridad Informática.
- Se implementó satisfactoriamente el agente AASI como herramienta de apoyo para detectar a tiempo las incidencias de seguridad informática en el Centro ISEC
- Se validó la solución implementada realizándose pruebas de unidad y de aceptación, arrojando resultados satisfactorios puesto que las deficiencias detectadas fueron solucionadas en iteraciones de prueba posteriores quedando al final el cliente satisfecho con la solución implementada.

Brindando el agente beneficios reales para el centro ISEC ya que automatiza gran parte del proceso de auditoría permitiendo identificar en un menor tiempo las violaciones a las políticas de seguridad informática de dicho Centro y facilita el trabajo de los auditores de seguridad informática ya que se ahorran la actividad de comprobar las políticas de seguridad que deben cumplir cada uno de los ordenadores del centro.

RECOMENDACIONES

Después de darle cumplimiento a los objetivos planteados en la investigación se recomienda:

- Implementar las funcionalidades que permitan enviar la información de las auditorías realizadas a SIGESI ya que ésta no se encontraba en funcionamiento durante el proceso de desarrollo.
- Implementar un mayor número de consultas a los sistemas operativos para comprobar más políticas que aumenten así la seguridad de la información.
- Brindar un mayor número de configuraciones que garanticen un software adaptable a otros entornos.
- Implementar un instalador de AASI que permita un mejor despliegue de la herramienta en las estaciones de trabajo.

REFERENCIAS BIBLIOGRÁFICAS

1. Márquez Pavone, Elisa. Tesis de la Universidad de Los Andes. Biblioteca Digital de SERBIULA. [En línea] septiembre de 2008. [Citado el: 22 de febrero de 2013.] http://tesis.ula.ve/pregrado/tde_arquivos/25/TDE-2012-09-26T04:17:47Z-782/Publico/marquezelisa.pdf.
2. Facultad de Contaduría y Ciencias Administrativas, Universidad Michoacana de San Nicolás de Hidalgo. [En línea] 2010. [Citado el: 20 de febreo de 2013.] http://www.fcca.umich.mx/descargas/apuntes/Academia_de_Informática/Administración_de_Unidades Informáticas R.C.M/UNIDAD V_2010.pdf
3. Cestero, José M. Tecnología Pyme. Auditorías de seguridad informática. [En línea] 26 de junio de 2009. [Citado el: 25 de febrero de 2013.] http://www.tecnologiapyme.com/tendencias/auditorías-de-seguridad-informática.
4. González Sánchez, José Luís. TDR Tesis Doctorales en Red. Universidades de Catalunya. [En línea] 24 de julio de 2001. [Citado el: 11 de marzo de 2013.] http://www.tdx.cat/bitstream/handle/10803/5966/09capitol6.pdf?sequence=9.
5. Labrador, Ramón M. Gómez. E.T.S de INGENIERIA INFORMÁTICA. Universidad de Sevilla. [En línea] 2013. [Citado el: 6 de marzo de 2013.] http://www.informática.us.es/~ramon/tesis/CORBA/Seminario-MASIF/.
6. Definición ABC. [En línea] [Citado el: 4 de marzo de 2013.] http://www.definicionabc.com/tecnología/seguridad-informática.php.
7. Mirabal Sarria, Yamilet y Maragoto Maragoto, Maria Isabel. PROPUESTA DE UNA GUÍA DE SEGURIDAD INFORMÁTICA INTEGRADA A LA GESTIÓN DE LA CALIDAD. Pinar del Rio, 2011.
8. Guindel Sánchez, Esmeralda y Ramos González, Miguel Ángel. CALIDAD Y SEGURIDAD DE LA INFORMACIÓN Y AUDITORÍA INFORMÁTICA. UNIVERSIDAD CARLOS III DE MADRID. Leganés, 2009.
9. SCD Servicios Informáticos S.R.L. SCD Servicios Informáticos. [En línea] 2012. [Citado el: 6 de marzo de 2013.] http://www.scd.com.ar/servicios/auditoría_informática.php.

10. Perito y Tasador. El Portal de Peritos Judiciales y Tasadores. [En línea] [Citado el: 6 de marzo de 2013.] http://www.peritoytasador.es/auditoría-de-seguridad-informática/.

11. Mirabal Sarria, Yamilet y Maragoto Maragoto, Maria Isabel. PROPUESTA DE UNA GUÍA DE SEGURIDAD INFORMÁTICA INTEGRADA A LA GESTIÓN DE LA CALIDAD. Pinar del Rio, 2011.

12. Jennings, Nick, Jennings, Nicholas R y Wooldridge, Michael J. Agent Technology: Foundations, Applications, and Markets.: Springer, 1998.

13. isoftland. [En línea] 2011. [Citado el: 8 de marzo de 2013.] http://www.isoftland.com/software/gfi/languard.

14. Microsoft. TechNet. [En línea] 2013. [Citado el: 13 de marzo de 2013.] http://technet.microsoft.com/es-es/security/cc184924.aspx.

15. Tenable Network Security. [En línea] 2013. [Citado el: 8 de marzo de 2013.] http://www.tenable.com/products/nessus.

16. Latindevelopers.com. [En línea] 2011. [Citado el: 8 de marzo de 2013.] http://www.latindevelopers.com/articulos/misc/instalar-ocs-inventory.php.

17. Carvajal Riola, Jose Carlos. Metodologías Ágiles: Herramientas y modelo de desarrollo para aplicaciones Java EE como metodología empresarial. 2008.

18. Beck, Kent. Extreme Programming Explained, Embrace Change.: Addison-Wesley, First Edition, 1999.

19. Definicion.de. [En línea] 2008. [Citado el: 25 de marzo de 2013.] http://definicion.de/lenguaje-de-programacion.

20. SQLite. [En línea] [Citado el: 1 de abril de 2013.] http://www.sqlite.org/about.html.

21. Rhee, Man Young. Cryptographic Principles: Algorithms and Protocols. Universidad Nacional de Seúl, República de Korea.

22. Pressman, Roger S. Ingeniería del Software: un enfoque práctico. Parte I y II. La Habana: Félix Varela, 2005.

23. Vignaga, Andrés y Perovich, Daniel. Facultad de Ingeniería. Universidad de la República-Uruguay. [En línea] [Citado el: 4 de abril de 2013.] http://www.fing.edu.uy/inco/grupos/coal/uploads/Investigación.

24. LARMAN, Craig. Applying UML and Patterns: Prentice Hall, 1998.

25. IEEE. [En línea] 1999. [Citado el: 8 de abril de 2013.] http://www.ieee.org

26. McBreen, Pete. Questioning Extreme Programming: Addison-Wesley, 2002.

BIBLIOGRAFÍA

1. Cudeiro Rodríguez, Díaz Martínez y Aguila Cudeiro. "La auditoría como una disciplina dentro de las ciencias contables. Enfoques teóricos y metodológicos de su praxis" en Observatorio de la Economía Latinoamericana, Nº 166, 2012.
2. Guindel Sánchez, Esmeralda y Ramos González, Miguel Ángel. *CALIDAD Y SEGURIDAD DE LA INFORMACIÓN Y AUDITORÍA INFORMÁTICA. UNIVERSIDAD CARLOS III DE MADRID.* Leganés, 2009.
3. Mirabal Sarria, Yamilet y Maragoto Maragoto, Maria Isabel. *PROPUESTA DE UNA GUÍA DE SEGURIDAD INFORMÁTICA INTEGRADA A LA GESTIÓN DE LA CALIDAD.* Pinar del Rio, 2011.
4. Beck, Kent. *Extreme Programming Explained, Embrace Change*: Addison-Wesley, First Edition, 1999.
5. Kent Beck. Extreme Programming Explained, Embrace Change, First Edition, Addison-Wesley 1999.
6. Chacón, Felipe Alexis Morales. Utilización de la metodología programación extrema en un caso práctico: “Sistema de Gestión y Control de Inspecciones Técnicas”. Talca, 2009.
7. Ivar Jacobson, Grady Booch, James Rumbaugh. EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE, Madrid, Addison Wesley, 2000.
8. Carvajal Riola, Jose Carlos. *Metodologías Ágiles: Herramientas y modelo de desarrollo para aplicaciones Java EE como metodología empresarial.* 2008.
9. Leonardo Gomez Contreras y Francisco González. Desarrollo de Proyectos de Software Basados en la metodología XP: Propuesta de una pauta de trabajo, 2005.
10. Carvajal Riola, José Carlos. *METODOLOGÍAS ÁGILES,* 2008.
11. Joskowicz, José. Reglas y Prácticas de Extreme Programming, 2008.
12. Echeverry Tobón, Luis M. and Delgado Carmona, Luz Elena. Caso Práctico de la Metodología Ágil al desarrollo de software, 2007.
13. Escrcibano, G. F. Introducción a Extreme Programming, 2002.
14. Jennings, Nick, Jennings, Nicholas R y Wooldridge, Michael J. *Agent Technology: Foundations, Applications, and Markets*: Springer, 1998.

15. Pressman, Roger S. *Ingeniería del Software: un enfoque práctico. Parte I y II.* La Habana: Félix Varela, 2005.

16. LARMAN, Craig. *Applying UML and Patterns*: Prentice Hall, 1998.

17. Rhee, Man Young. *Cryptographic Principles: Algorithms and Protocols.* Universidad Nacional de Seúl, República de Korea.

18. Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, 2005.

19. McBreen, Pete. *Questioning Extreme Programming*: Addison-Wesley, 2002.

20. Franklin Rivero Duharte. Pruebas Unitarias de Software en la Plataforma J2EE, Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas, La Habana, 2008.

21. Facultad de Contaduría y Ciencias Administrativas, Universidad Michoacana de San Nicolás de Hidalgo. [En línea] 2010. [Citado el: 20 de febreo de 2013.] http://www.fcca.umich.mx/descargas/apuntes/Academia_de_Informática/Administración_de_Unidades Informáticas R.C.M/UNIDAD V_2010.pdf.

22. SISMAN. Universidad Técnica de Manabí. [En línea] 2013. [Citado el: 22 de febrero de 2013.] http://www.sisman.utm.edu.ec/libros/FACULTAD_DE_CIENCIAS_INFORMÁTICAS/CARRERA_DE INGENIERIA DE SISTEMAS INFORMÁTICOS/10/Auditoría Informática/aui1.pdf.

23. Camargo Montero, Daniel. Tesis Digitales. Universidad de las Américas Puebla. [En línea] 2 de febrero de 2007. [Citado el: 22 de febrero de 2013.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/camargo_m_d/.

24. Márquez Pavone, Elisa. Tesis de la Universidad de Los Andes. Biblioteca Digital de SERBIULA. [En línea] septiembre de 2008. [Citado el: 22 de febrero de 2013.] http://tesis.ula.ve/pregrado/tde_arquivos/25/TDE-2012-09-26T04:17:47Z-1782/Publico/marquezelisa.pdf.

25. Cestero, José M. Tecnología Pyme. Auditorías de seguridad informática. [En línea] 26 de junio de 2009. [Citado el: 25 de febrero de 2013.] http://www.tecnologiapyme.com/tendencias/auditorías-de-seguridad-informática.

26. Sánchez, Esmeralda Guindel. e-archivo. Universidad Carlos III de Madrid. [En línea] 23 de noviembre de 2009. [Citado el: 26 de febrero de 2013.] http://e-archivo.uc3m.es/bitstream/10016/8510/1/proyectoEsmeralda.pdf.

27. Rodríguez, Germán Ramírez. SISMAN. Universidad Tecnica de Manabi. [En línea] 2009. [Citado el: 5 de marzo de 2013.] http://www.sisman.utm.edu.ec/libros/FACULTAD DE CIENCIAS ADMINISTRATIVAS Y ECONÓMICAS/CARRERA DE CONTABILIDAD Y AUDITORÍA/09/auditoría de sistemas informáticos/SOBRE LA AUDITORÍA INFORMÁTICA Y LOPD D.

28. Labrador, Ramón M. Gómez. E.T.S de INGENIERIA INFORMÁTICA. Universidad de Sevilla. [En línea] 2013. [Citado el: 6 de marzo de 2013.] http://www.informática.us.es/~ramon/tesis/CORBA/Seminario-MASIF/.

29. SCD Servicios Informáticos S.R.L. SCD Servicios Informáticos. [En línea] 2012. [Citado el: 6 de marzo de 2013.] http://www.scd.com.ar/servicios/auditoría_informática.php.

30. Perito y Tasador. El Portal de Peritos Judiciales y Tasadores. [En línea] [Citado el: 6 de marzo de 2013.] http://www.peritoytasador.es/auditoría-de-seguridad-informática/.

31. isoftland. [En línea] 2011. [Citado el: 8 de marzo de 2013.] http://www.isoftland.com/software/gfi/languard.

32. Tenable Network Security. [En línea] 2013. [Citado el: 8 de marzo de 2013.] http://www.tenable.com/products/nessus.

33. Latindevelopers.com. [En línea] 2011. [Citado el: 8 de marzo de 2013.] http://www.latindevelopers.com/articulos/misc/instalar-ocs-inventory.php.

34. Kent Beck, Manifesto for Agile Software Development. [En línea] [Citado el: 11 de abril de 2013.] http://www.agilemanifesto.org/

35. González Sánchez, José Luís. TDR Tesis Doctorales en Red. Universidades de Catalunya. [En línea] 24 de julio de 2001. [Citado el: 11 de marzo de 2013.] http://www.tdx.cat/bitstream/handle/10803/5966/09capitol6.pdf?sequence=9.

36. Microsoft. TechNet. [En línea] 2013. [Citado el: 13 de marzo de 2013.] http://technet.microsoft.com/es-es/security/cc184924.aspx.

37. James Donovan Wells. Extreme Programming: A gentle introduction. [En línea] 2009 [Citado el: 20 de marzo de 2013.] http://www.extremeprogramming.org

38. Miguel Jaque Barbero. Ventajas de la programación extrema en el entorno empresarial. DATA TI. [En línea] 2009. [Citado el: 20 de marzo de 2013.] http://www.ilkebenson.com/articulos/xp.pdf.

39. Julio Ariel Hurtado, Cecilia Bastarrica. PROYECTO SIMEP-SW1 Trabajo de Investigación: "Hacia una Línea de Procesos Ágiles Agile SPsL", Universidad del Cauca. [En línea] 2009 [Citado el: 21 de marzo de 2013.] http://www.dcc.uchile.cl/TR/2005/TR_DCC-2005-008.pdf.

40. Netbeans.org. [En línea] [Citado el: 1 de abril de 2013.] http://netbeans.org/community/releases/69/relnotes_es.html.

41. PostgreSQL. [En línea] [Citado el: 4 de abril de 2013.] http://www.postgresql.org/about/.

42. SQLite. [En línea] [Citado el: 4 de abril de 2013.] http://www.sqlite.org/about.html.

43. SQLite Latino America. [En línea] [Citado el: 4 de abril de 2013.] http://sqlite-latino.blogspot.com/.

44. Visual Paradigm. [En línea] [Citado el: 8 de abril de 2013.] http://www.visual-paradigm.com.

45. Rojas, Diego. IComparable. [En línea] [Citado el: 22 de abril de 2013.] http://icomparable.blogspot.com/2008/10/arquitectura-n-tier-o-arquitectura-n.html.

46. Vignaga, Andrés y Perovich, Daniel. Facultad de Ingeniería. Universidad de la República-Uruguay. [En línea] [Citado el: 22 de abril de 2013.] http://www.fing.edu.uy/inco/grupos/coal/uploads/Investigación.