Universidad de las Ciencias Informáticas
Facultad 3

Sistema informático para la realización de auditorías de Seguridad Informática en CEGEL

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

Autor:

Verónica Hernández Urgelles

Tutores:

Ing. Yordanis García Leiva

Ing. Juan David Gómez Amador

Co-Tutor:

Ing. Reinier Silverio Figueroa

Junio, 2016

"Año 58 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____

Verónica Hernández Urgelles

_____ _____

Ing. Yordanis García Leiva Ing. Juan David Gómez Amador

_____

Ing. Reinier Silverio Figueroa

AGRADECIMIENTOS

Primeramente, agradezco a Dios por estar siempre a mi lado y no abandonarme en los malos momentos.

A todos los que de una forma u otra tuvieron que ver con mi educación. En primer lugar, agradezco a mi familia en general, por ser constante veladora de mis largos años de estudios.

En especial a mi madre por todo su amor, protección, cuidados y apoyo incondicional durante tantos años.

A mi padre, que, a pesar de la distancia, siempre ha aportado su granito para contribuir a mi desarrollo como persona y como mujer. Gracias por tus consejos.

A mis abuelitas Caridad y Esperanza por ser las mejores abuelas del mundo. Gracias por los años de amor incondicional, los mimos, los consejos y por estar siempre ahí para mí cuando más las necesito. A mi abuelito Pedrín que, aunque no se encuentra entre nosotros, sé que está orgulloso de mí.

A mi hermana Beli, a mis primas Sandra, Jade, Samantha y a mi primo Migue por hacer mi vida diferente y divertida.

A mis tías Ledy y Mely por todo su apoyo y su cariño.

A mis súper amigas más que amigas hermanas Yamina, Daylis, Mara, Zule y Yane, por todos los mensajes de aliento y todos los momentos geniales que paso con ustedes.

A mis súper compañeras de apartamento Lissette (Mimi), Esther, Leidy y Dania por su comprensión, su ayuda, su aguante y por su amistad y apoyo en los momentos malos, buenos, regulares. Son las mejores.

A todos mis amigos y amigas aquí en la UCI, los que vienen desde 1ro y a los que he conocido en mi paso por la universidad por ser seres humanos maravillosos y permitirme por un tiempo formar parte de sus vidas.

A mis tutores por estar siempre cuando los necesitaba y por su gran apoyo, por su ayuda personal y profesional en la creación de este trabajo, por el tiempo dedicado, por soportarme, por la experiencia y los consejos, ya que sin su apoyo y dedicación este trabajo no se hubiera realizado. Gracias a los dos, son seres humanos especiales, principalmente Yordanis.

A todos los que de una forma u otra me han ayudado con la tesis, en especial a Yoandris y Yordanis Crespo, que se portaron súper especial conmigo. Gracias por su apoyo y amistad.

A todos aquellos que me ayudaron a realizar este sueño, muchas gracias.

A todos en general.

DEDICATORIA

A todos los que me quieren y creen en mí, va dedicado este trabajo, en especial a mi familia, gracias por su apoyo, y a los que no también, pues me han enseñado a ser más fuerte y a confiar en mí misma.

RESUMEN

La seguridad informática está compuesta por el conjunto de métodos y herramientas destinados a proteger los activos informáticos de una institución y tiene como principios básicos, garantizar la confidencialidad, integridad y disponibilidad de la información. Entre las actividades que se realizan con el fin de comprobar la implementación de las medidas y procedimientos necesarios que certifican el cumplimiento de estos principios, se encuentran las auditorías al sistema de seguridad informática de cada entidad. En la actualidad existen un conjunto de soluciones informáticas que agilizan el desarrollo de estas actividades, sin embargo, no todas se adaptan a las comprobaciones que requiere hacer cada entidad durante la realización de una auditoría. La presente investigación tiene como objetivo, desarrollar una aplicación informática para la realización de auditorías de seguridad informática en el Centro de Gobierno Electrónico. La solución propuesta permite a la dirección del centro tener de forma periódica, un reporte con el estado del cumplimiento de las políticas de seguridad informática en cada computadora del área conectada a la red. Esta fue implementada utilizando tecnologías de código abierto y constituye una herramienta de apoyo para el grupo de activistas de seguridad informática en el Centro de Gobierno Electrónico.

PALABRAS CLAVES: auditoría**,** políticas**,** reportes**,** seguridad informática

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con la evolución de las Tecnologías de la Información y las Comunicaciones (TIC), el desarrollo de la informática ha desempeñado un papel importante en las diferentes esferas de la sociedad. El uso frecuente de las TIC ha hecho necesario, la implementación de medidas y políticas de seguridad, encargadas de velar por la protección de los medios. La Seguridad Informática (SI) se logra mediante la implementación de un grupo de controles que incluyen políticas, normas, métodos, técnicas, procedimientos, estructuras organizativas y sistemas de hardware y software, destinados a garantizar un sistema informático seguro y confiable, en función de preservar los principios de confidencialidad, integridad y disponibilidad de la información [1].

- Confidencialidad: la información o los activos informáticos son accedidos solo por las personas autorizadas para hacerlo [1].
- Integridad: los activos o la información solo pueden ser modificados por las personas autorizadas y de la forma autorizada [1].
- Disponibilidad: los activos informáticos son accedidos por las personas autorizadas en el momento requerido [1].

Con el propósito de velar por la puesta en práctica de las normas y procedimientos que en materia de SI las entidades deben cumplir, se requiere de la planificación periódica de auditorías a su sistema de SI. Las auditorías de SI permiten la realización de revisiones exhaustivas al sistema de SI de una entidad, con el fin de controlar el cumplimiento de cada una de las políticas de seguridad informática (PSI) definidas para esta. El desarrollo de estas actividades tiene como principal objetivo, velar por la protección de las estaciones de trabajo, redes de comunicaciones, servidores y recursos humanos presentes en cada institución. Las auditorías analizan los procesos relacionados únicamente con la seguridad, esta puede ser física, lógica o locativa, pero siempre orientada a la protección de la información.

La Universidad de las Ciencias Informáticas (UCI), constituye un ejemplo en el empleo de los avances tecnológicos en la rama de la informática, sin descuidar la seguridad y su supervisión periódica. Teniendo en cuenta lo anterior, esta institución tiene definidas PSI encaminadas a garantizar el uso correcto de las TIC, controladas a través de la realización de auditorías al sistema de SI de cada área de la universidad.

El Centro de Gobierno Electrónico (CEGEL), adjunto a la Facultad 3 de la UCI, cuenta con un grupo de activistas distribuidos por laboratorios y un asesor de seguridad informática, responsabilizados de controlar el cumplimiento de las PSI establecidas por la universidad y las propias del área, a través de la realización de auditorías. Entre las principales PSI definidas se encuentran:

- Mantener las computadoras en el dominio uci.cu
- Tener instalado, configurado y actualizado el antivirus Kaspersky en cada una de las computadoras
- Tener activado el corta fuegos en todas las máquinas
- No tener carpetas compartidas
- Sistema operativo actualizado

El control en CEGEL de las PSI antes mencionadas se realiza a través del uso de una aplicación que no satisface el proceso de auditorías al sistema de SI en el área, al presentar las siguientes deficiencias: hay que ejecutarla de forma independiente en cada máquina. Los datos generados deben ser procesados de forma manual, lo que dificulta la rápida obtención de los resultados y no brinda la posibilidad de generar reportes con el estado de las PSI en comprobaciones anteriores, que posibiliten contar con una trazabilidad en el desarrollo del proceso, es decir, tener un historial de las comprobaciones realizadas en cada una de las máquinas en el tiempo. Además, existen algunos casos de estaciones de trabajo con sistema operativo Linux, donde la PSI (dominio uci.cu) no es identificada por la aplicación, a pesar de estar configurada la misma en las máquinas auditadas.

En la UCI también existe la herramienta para la Gestión de Recursos de Hardware y Software (GRHS), la cual a pesar de tener entre sus funciones el control de las PSI en cada máquina, hasta el momento solo permite el control de dos de estas (dominio uci.cu y el antivirus con sus configuraciones) y tampoco genera reportes con el historial del estado de las PSI en otras revisiones, que posibiliten contar con una trazabilidad en el desarrollo del proceso. Por tal motivo esta herramienta tampoco satisface el proceso de auditorías al sistema de SI en CEGEL.

La **problemática** antes descrita permite identificar el siguiente **problema de investigación:**

El desarrollo actual de las auditorías de SI en el Centro de Gobierno Electrónico no garantiza trazabilidad y agilidad en el desarrollo del proceso.

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio:** Seguridad Informática.

Para ello se identifica como **campo de acción:** aplicaciones informáticas para el control de Políticas de Seguridad Informática.

Determinándose como **objetivo general:** desarrollar un sistema informático para la realización de auditorías de SI en el Centro de Gobierno Electrónico, que garantice trazabilidad y agilidad en el proceso.

Se desglosan del objetivo general los siguientes **objetivos específicos:**

- ✓ Elaborar el marco teórico referencial de la investigación relacionado con el proceso de auditorías de SI.
- ✓ Implementar el sistema para la realización de auditorías de SI en el Centro de Gobierno Electrónico.
- ✓ Validar el correcto funcionamiento de la aplicación y que dé cumplimiento al objetivo general de la investigación.

Determinándose como **idea a defender:** el desarrollo de un sistema informático para la realización de auditorías de SI en el Centro de Gobierno Electrónico garantizará trazabilidad y agilidad en el proceso.

Métodos Teóricos:

Histórico-Lógico: permitió realizar un estudio sobre las principales tendencias con respecto a las herramientas para la realización de auditorías de SI existentes, metodologías de desarrollo y la forma en que se brindan determinados servicios relacionados con la SI y las auditorías tanto en el mundo, como en el marco productivo de la universidad, con el fin de seleccionar las más apropiadas para el desarrollo del sistema.

Analítico-Sintético: permitió realizar el estudio teórico de la investigación facilitando el análisis de documentos y fuentes bibliográficas para la extracción de los elementos más importantes acerca del proceso de desarrollo del sistema.

Inductivo-Deductivo: la inducción permitió realizar un razonamiento partiendo de lo particular a lo general, reflejando en el resultado final lo común que existe en cada bibliografía estudiada. La deducción brindó la posibilidad a través de los conocimientos

generales adquiridos durante la investigación, inferir otros conocimientos lógicos como las conclusiones parciales y generales de la tesis.

Métodos Empíricos:

Entrevista: fue utilizado con el objetivo de definir y comprender las necesidades del cliente, capturar los requisitos correspondientes al sistema y obtener información que apoye la realización de la investigación.

Observación: se utilizó a través del estudio realizado a herramientas informáticas que existen en el mundo que permiten realizar auditorías de SI, para determinar los elementos más comunes que están presentes en las mismas.

Medición: permitió medir la calidad de la especificación de los requisitos y el grado de ambigüedad de estos, además de obtener una medida de la calidad del diseño para su validación.

El presente trabajo está compuesto por 3 capítulos los cuales se estructuran de la siguiente forma:

Capítulo 1: Fundamentación Teórica.

En el presente capítulo se definen los fundamentos teóricos de la investigación, exponiéndose los principales conceptos relacionados con el objeto de estudio, para una mejor compresión del tema. También se realiza un análisis sobre las experiencias desarrolladas a nivel nacional e internacional, de las cuales se exponen sus principales características y desventajas que demuestran por qué no son utilizadas, y hacen necesaria la realización de la presente tesis. Además, se describen los elementos fundamentales de la metodología, herramientas y tecnologías a utilizar para el desarrollo del sistema informático propuesto.

Capítulo 2: Análisis, diseño e implementación del sistema.

En este capítulo se realiza una descripción de las principales características del Sistema para la realización de Auditorías de Seguridad Informática para CEGEL, así como del diseño del mismo. Se definen los requisitos funcionales y no funcionales capturados dadas las necesidades del cliente para la implementación del sistema, partiendo del estudio del estado del arte realizado en el capítulo anterior. También se definen otros elementos

importantes dentro del proceso de desarrollo, tales como: historias de usuario, arquitectura, patrones empleados y elementos de la implementación del mismo.

Capítulo 3: Validación del sistema.

En este capítulo se realizan, como define la metodología XP, pruebas unitarias y pruebas de aceptación en conjunto con el cliente al sistema propuesto, describiéndose los principales artefactos generados, como es el caso de las actas de libración interna de productos de software y de aceptación del cliente. También se expone la descripción de los casos de prueba diseñados para el sistema. Así como los resultados obtenidos en las pruebas realizadas al sistema propuesto y las métricas aplicadas a los requisitos y diseño de las clases.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se realiza un análisis sobre las experiencias desarrolladas a nivel nacional e internacional, de las cuales se exponen sus principales características y desventajas que demuestran por qué no son utilizadas, y hacen necesario la realización de la presente tesis. Además, se describen los elementos fundamentales de la metodología, herramientas y tecnologías a utilizar para el desarrollo del sistema informático propuesto.

1.2 Tendencias actuales

Debido que el tema de la SI es de suma importancia en el mundo, existen una gran cantidad de herramientas que contribuyen al mantenimiento de la misma, pero muy pocas son utilizadas para la realización de auditorías de SI, pues no todas están implementadas para realizar específicamente esta función. Con el propósito de adquirir experiencias, se analizaron algunas de estas herramientas, las más utilizadas a escala nacional e internacional, en busca de características similares que sirvan para guiar el diseño del sistema informático propuesto.

Nessus (The Nessus Security Scanner): es un auditor de seguridad remoto; es el escáner de vulnerabilidades y evaluaciones de configuración más utilizado en el mundo, no solo por su gran velocidad a la hora de realizar sus descubrimientos, por el tiempo que lleva en el mercado (la primera versión de Nessus es de 1997) o por su modelo cliente-servidor que permite ejecutar los clientes de Nessus en equipos con menos prestaciones (como dispositivos móviles), sino porque constantemente se está renovando y adaptando a las necesidades actuales de los profesionales de la seguridad informática. Hace posible evaluar módulos de seguridad intentando encontrar puntos vulnerables que deberían ser reparados. Permite generar reportes en HTML (Lenguaje de Marca de Hipertexto), XML (Lenguaje de Marca Extensible) , LaTeX, y texto ASCII; también sugiere soluciones para los problemas de seguridad. Está compuesto por dos partes: un servidor y un cliente. El servidor/daemon, “nessusd” se encarga de los ataques, mientras que el cliente “nessus”, se ocupa del usuario por medio de una interfaz. Nessus previene los ataques de red mediante la identificación de las vulnerabilidades y problemas de configuración que utilizan los hackers para penetrar en la red. Desarrollado por *Tenable Network Security*, es

multiplataforma, bajo licencia GPL (Licencia pública de GNU) y su última versión estable es la 5.0.1 [6].

Nmap (Network Mapper): es de utilidad gratuita y de código abierto, es una herramienta para la detección de redes y auditoría de seguridad. A muchos administradores de red les resulta útil para realizar tareas tales como inventario de la red, los horarios de actualización de servicio de gestión, monitoreo o el tiempo de servicio del host. *Nmap* utiliza paquetes IP en nuevas formas de determinar qué servicios están disponibles en la red de ordenadores, qué servicios (nombre de la aplicación y la versión) los hosts están ofreciendo, los sistemas operativos (y versiones del sistema operativo) que se están ejecutando, qué tipo de filtros de paquetes/cortafuegos están en uso, y otras características. Fue diseñado para escanear rápidamente grandes redes, pero funciona bien contra los hosts individuales. Nmap se ejecuta en todos los principales sistemas operativos, y paquetes binarios oficiales que están disponibles para Linux, Windows y Mac OS X. La suite Nmap incluye un visor de interfaz gráfica de usuario avanzada y resultados (*Zenmap*), transferencia de datos flexible, la redirección, y herramienta de depuración (*Ncat*), utilidad para la comparación de los resultados del análisis (*Ndiff)*, y una herramienta de análisis de generación de paquetes y la respuesta (*Nping*). Fue desarrollado por *Fyodor*, es multiplataforma, bajo GLP y su última versión estable es la 7.0.0 [7].

Gestor de Recursos de Hardware y Software (GRHS): es una herramienta creada en la UCI con el fin de tener un mejor control de los recursos de hardware y software de una red de computadoras; por lo que, en coordinación con el Centro de Telemática de dicha universidad, deciden desarrollar una aplicación que automatice la mayor parte del proceso. Es una aplicación cliente - servidor cuyo objetivo principal es centralizar en el servidor la información del hardware y el software instalados en las estaciones clientes. La aplicación cliente se encarga de obtener la información de hardware y software instalados y envía los datos recolectados al servidor, determinando si hubo cambios en el hardware o el software a partir del inventario obtenido anteriormente. El sistema agente, así como el sistema servidor son propiedad exclusiva de la universidad [8].

Para un mejor análisis de las herramientas estudiadas y basándose en algunas de las PSI antes mencionadas. Se realizó una comparación entre las herramientas, para observar de

forma más detallada, si alguna garantiza una mejora en el proceso de realización de auditorías en CEGEL. En la tabla 1 se ilustra el resultado de esta comparación.

Tabla 1. Comparación entre las herramientas analizadas.

Herramientas investigadas	Criterios					
	Políticas de Seguridad					Reportes
	Dominio uci.cu	Carpetas compartidas	Antivirus instalado y activado	Corta fuegos activado	Sistema Operati-vo actualiza-do	Genera reportes
Nessus	-	-	-	-	-	X
Nmap	-	-	-	X	X	-
GRHS	X	-	X	-	-	-

Luego del análisis de algunas de las herramientas, utilizadas a nivel internacional y nacional, para la realización de auditorías de SI. Se llegó a la conclusión que a pesar de todas tener el mismo fin, controlar la Seguridad Informática, no permiten controlar gran parte de las PSI establecidas por la universidad, debido a que son herramientas muy diferentes y poseen variadas funcionalidades. A partir de lo anterior, se determina que ninguna de las herramientas analizadas responde al problema de la presente investigación, razón por la cual, se decide desarrollar el sistema propuesto.

1.3 Metodología de Desarrollo de Software

Una metodología de desarrollo de software está compuesta por un conjunto de procedimientos, técnicas, herramientas y artefactos, que facilitan el trabajo de los desarrolladores. Existen dos grupos de metodologías de desarrollo de software, las ágiles y las tradicionales [9].

En la presente investigación se decidió utilizar una metodología ágil, debido a que la solución propuesta representa un software pequeño, de corta duración y con un reducido número de miembros. Las metodologías ágiles constituyen una serie de técnicas para la gestión de proyectos, surgidas como contraposición a los métodos clásicos de gestión.

Están orientadas a guiar el proceso de desarrollo de proyectos pequeños y con equipos de trabajos con personal reducido. Constituyen una solución a corto plazo, proponen mejorar la calidad del software, teniendo como premisa la comunicación inmediata y directa con el cliente. Tienen como principal punto la adaptación a cualquier cambio en un proyecto, lo cual se hace con el objetivo de aumentar las posibilidades de éxito [10].

A continuación, la tabla 2 representa algunas de las metodologías ágiles utilizadas a nivel nacional e internacional, realizando una comparación entre las mismas, teniendo en cuenta un conjunto de características definidas para el análisis:

Tabla 2. Metodologías ágiles de desarrollo de software.

Características	XP	Scrum	Crystal	SXP
Tamaño de proyecto	Pequeños y medianos	De cualquier tamaño con entornos complejos	De cualquier tamaño dependiendo del método de la familia	Pequeños y medianos
Tamaño de equipo de trabajo	Equipos pequeños (menos de 20 miembros)	Múltiples tamaños	Múltiples tamaños dependiendo del método de la familia	Pequeños equipos de trabajo
Tiempo de entrega	Una entrega no debería tardar más de un mes	Se necesitan los resultados pronto	Entregas frecuentes (desde semanales hasta trimestrales)	Programación rápida o extrema
Comunicación dentro del equipo y con el cliente	Su objetivo es el trabajo en equipo incluyendo al cliente	Reunión diaria de 15 minutos de equipo de desarrollo	Informal (cara a cara)	Trabajo en equipo, incluye al cliente
Flexibilidad a cambios	Mayor énfasis en la adaptabilidad y satisfacción del cliente	Los requisitos son cambiantes y pocos definidos	Adaptabilidad y satisfacción del cliente	Cubrir las expectativas del cliente
Documentación de proyecto	Documentación básica	Documentación básica	Documentación básica	Documentación básica

Estilo de desarrollo	Iterativo, rápido e incremental	Iterativo, rápido e incremental	Incremental	Iterativo, rápido e incremental

Atendiendo al corto período de tiempo, y a la cantidad de integrantes del proyecto (conformado solo por un programador y un cliente), se determina utilizar la metodología *Extreme Programming* (XP). Ya que la misma permite estructurar, planear y controlar el proceso de desarrollo del producto a implementar. La metodología XP responde a las necesidades de la investigación y a las condiciones de trabajo. Esta metodología permite incrementar considerablemente la productividad del equipo de desarrollo, conseguir con más facilidad la satisfacción del cliente, mantener los estándares de calidad y cumplir los plazos con mayor exactitud y consistencia.

1.4 Herramientas de Ingeniería del Software Asistidas por Computadoras

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE) se definen, según el autor Ian Sommerville, como “el software que se utiliza para ayudar a las actividades del proceso del software como la ingeniería de requerimientos, el diseño, el desarrollo de programas y las pruebas. Las herramientas CASE incluyen editores de diseño, diccionarios de datos, compiladores, depuradores, herramientas de construcción de sistemas” [11]. Existen diversas herramientas CASE, como es el caso de: Erwin, *EasyCASE*, Oracle Designer y Visual Paradigm, siendo esta última una de las más utilizadas para el desarrollo de sistemas en los proyectos productivos de la UCI, siendo seleccionada para el desarrollo de la presente investigación.

1.4.1 Herramienta de modelado: Visual Paradigm

“Visual Paradigm es una herramienta CASE de modelado visual que facilita la construcción de artefactos en un proceso de desarrollo de software mediante el Lenguaje de Modelado Unificado (UML). Soporta una amplia gestión de casos de uso y diseño de base de datos relacionales y proporciona medidas más eficaces en el análisis y diseño de sistemas. Algunas de las funcionalidades que brinda la herramienta son la generación de código y de base de datos a partir de los diagramas UML realizados, la realización de ingeniería inversa, generación automática de informes en formato PDF, Word o HTML; generación de máquinas de estados, integración con Visio y Rational Rose” [12].

Teniendo en cuenta la existencia de una Licencia UCI para el uso de Visual Paradigm y las características de esta herramienta, para el desarrollo del presente trabajo se selecciona la misma en su versión 10.1. debido a que está diseñada para una amplia gama de usuarios, incluyendo los ingenieros de software, analistas de sistemas, analistas de negocios y arquitectos de sistemas, o para cualquier persona que esté interesada en la construcción de sistemas de software que utilizan un enfoque orientado a objetos. Además, es compatible con los últimos estándares de la notación UML [12].

1.4.2 Lenguaje de modelado UML

El Lenguaje de Modelado Unificado (UML), es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, respaldado el mismo por el OMG (siglas en inglés de *Object Management Group*). UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases y objetos, hasta la implementación y configuración con los diagramas de despliegue [13]. En el presente trabajo se hace uso de este lenguaje, en su versión 2.0.

1.5 Lenguajes de programación

Un lenguaje de programación es un “conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Un lenguaje de programación permite a uno o más programadores especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias” [14].

Existen disímiles lenguajes de programación que propician la implementación de aplicaciones: web, de escritorio o para móviles. Entre los lenguajes para el desarrollo de aplicaciones de escritorio se destacan C#, C, C++ y Java. Por los beneficios que brinda, las características del mismo y las exigencias del cliente, se decide utilizar Java como lenguaje de programación para la implementar el sistema informático propuesto.

1.5.1 Java

Java es un lenguaje totalmente orientado a objeto, diseñado como una mejora de C++. Posee una curva de aprendizaje muy rápida. Proporciona una colección de clases para su uso en aplicaciones de red, que permite establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas. Es un lenguaje robusto ya que fue diseñado para crear softwares altamente fiables. Java está diseñado para soportar aplicaciones que serán ejecutados en los más variados entornos de red, desde Unix a Windows, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos [15].

1.6 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado, llamado también IDE (sigla en inglés de *Integrated Development Environment*), es un programa informático compuesto por un conjunto de herramientas de programación que puede dedicarse exclusivamente a un solo lenguaje de programación o bien puede utilizarse para varios [16]. Existen diversos IDE, algunos de los más utilizados a nivel mundial son: Microsoft Visual Studio, Eclipse y NetBeans IDE. Para el desarrollo de la presente investigación se decide utilizar NetBeans.

1.6.1 NetBeans IDE

Herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. La plataforma permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados (módulos). Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. NetBeans IDE permite desarrollar aplicaciones Java en entorno de escritorio, móviles y aplicaciones web. Está escrito y pensado para Java; pero brinda soporte para los lenguajes de programación: C/C++, Ruby, Phyton, HTML, CSS (Hoja de Estilo de Cascada) y PHP (Procesador de Hipertexto). Es gratuito y de código abierto [17]. Por todo lo antes expuesto es que se escoge esta herramienta, como IDE para el desarrollo del Sistema informático para la realización de auditorías de SI.

1.7 Marco de trabajo

Un marco de trabajo es un entorno, plataforma o estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Los marcos de trabajo facilitan el desarrollo de software, permiten evitar los detalles de bajo nivel, concentrando más esfuerzos y tiempo en identificar los requerimientos del software [18].
Existen un conjunto de marcos de trabajo que se utilizan para el desarrollo de aplicaciones en Java, ejemplo: Hibernate, JDBC, Spring y XEGFORT. La autora del presente trabajo decide utilizar este último para el desarrollo del sistema propuesto en la presente investigación. Teniendo en cuenta que el marco de trabajo XEGFORT constituye la base para el desarrollo Java en CEGEL y se perfila a convertirse en la arquitectura de referencia de Java a nivel de universidad.

1.7.1 XEGFORT

Este marco de trabajo está diseñado para la creación de aplicaciones con arquitectura Cliente-Servidor utilizando la plataforma Edición Empresarial de Java (JEE por sus siglas en inglés). Consta de dos partes, una que se encuentra en el lado del cliente facilitando la creación de la capa de presentación y la comunicación con los objetos remotos desplegados en el servidor y una segunda que se encuentra en el lado del servidor donde se encuentra la lógica de negocio. XEGFORT provee un alto nivel de configurabilidad y adaptabilidad, lo cual lo convierte en un software altamente reusable. Actualmente este marco de trabajo tiene grandes condiciones para la administración de las aplicaciones informáticas [19].
Algunas de las funcionalidades y componentes que provee [19]:

- Una interfaz de usuario básica
- Un mecanismo para la internacionalización de las interfaces de usuario.
- Un mecanismo para el acceso a los procedimientos remotos publicados en un servidor de aplicaciones.
- Un mecanismo de mensajería.
- Un mecanismo para gestionar las excepciones lanzadas por la aplicación que lo utilice.

1.8 Sistema Gestor de Bases de Datos

Un Sistema Gestor de Base de Datos (SGBD) es una colección de programas que permite crear y mantener una base de datos, en el cual se pueden almacenar datos de manera estructurada, organizada y con la menor redundancia posible. Posibilita a su vez, la integridad, confidencialidad y seguridad de la información. Estos sistemas se utilizan como interfaz entre las bases de datos y el usuario. El propósito general de los SGBD es el de manejar de manera clara, sencilla y ordenada un conjunto de datos [20]. Son diversos los SGBD existentes, destacándose: MySQL, Oracle, Microsoft SQL Server, Microsoft Access, Apache Derby, SQLite y PostgreSQL, siendo este último el escogido para el desarrollo del Sistema informático para la realización de auditorías de SI, debido a que permite obtener una respuesta rápida al consultar su contenido y posee gran capacidad de almacenamiento dependiendo solo del espacio disponible del disco duro.

1.8.1 PostgreSQL

PostgreSQL es un SGBD objeto-relacional, distribuido bajo la licencia Berkeley Software Distribution (BSD) y cuenta con su código fuente disponible libremente. Es uno de los SGBD de código abierto más potente del mercado comparado con otros sistemas comerciales como Oracle y MySQL. PostgreSQL, funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema [21].

La selección del SGBD PostgreSQL en su versión 9.4.1 para el desarrollo del sistema informático para la realización de auditorías de SI, se debe al resultado de la comparación con los SGBD MySQL y Oracle. Los cuales tienen desventajas, pues poseen licencia propietaria en el uso comercial y personal. Igualmente se analiza el gestor de base de datos Firebird que posee licencia libre, pero tiene varias limitantes con respecto a PostgreSQL en cuanto a tamaño de la base de datos, límite de tablas, entre otras.

1.9 Métricas de validación de diseño

La serie de métricas LK (propuestas por Lorenz y Kidd) son las seleccionadas para la validación del diseño de las clases de la solución, de ellas específicamente la métrica Tamaño Operacional de las Clase (TOC) que consiste en medir el tamaño general de una clase tomando como valores el total de operaciones y el número de atributos (operaciones y atributos tanto heredados como privados de la instancia), encapsulados por la clase; y la métrica Relaciones entre Clases

(RC) cuyo resultado viene dado por el número de relaciones de uso de una clase con otra u otras.

A través de estas métricas se puede medir el estado de los atributos de calidad que a continuación se mencionan:

- **Responsabilidad:** consiste en la responsabilidad asignada a una clase modelada de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización:** consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras.
- **Complejidad de mantenimiento:** consiste en el grado de esfuerzo necesario a realizar para desarrollar una reparación, una mejora o una corrección de algún error de un diseño de software.
- **Cantidad de pruebas**: consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.

1.9.1 Métrica Relaciones entre Clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra y evalúa los atributos de calidad, acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas. A continuación, se explican los pasos para aplicar la métrica:

1. Determinar la cantidad de relaciones de uso (CRU) que poseen las clases a medir.
2. Calcular el promedio de las CRU.
3. Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 3.

Tabla 3. Rango de valores para medir la afectación de los atributos de calidad (RC)**.**

Atributos de calidad	Clasificación	Criterio
Acoplamiento	Ninguna	CRU = 0
	Baja	CRU = 1
	Media	CRU = 2
	Alta	CRU > 2
Complejidad de mantenimiento	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio
Reutilización	Baja	CRU > 2* promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU <= Promedio
Cantidad de pruebas	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio

1.9.2 Métrica Tamaño Operacional de las Clase (TOC)

La métrica TOC se aplicada a cada una de las clases del diseño con el objetivo de medir la calidad de las mismas con respecto a su grado de responsabilidad, complejidad de implementación y reutilización. A continuación, se explican los pasos que se llevaron a cabo para aplicar la métrica:

1. Cálculo del umbral. El umbral se toma del tamaño general de una clase que se determina sumando todas las operaciones que posee.
2. Calcular el promedio de los umbrales.
3. Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 4.

Tabla 4. Rango de valores para medir la afectación de los atributos de calidad (TOC).

Atributos de calidad	Clasificación	Criterio
Responsabilidad	Baja	Umbral<= Promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral > 2* promedio
Complejidad de mantenimiento	Baja	Umbral <= Promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral > 2* promedio
Reutilización	Baja	Umbral > 2* promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral <= Promedio

1.10 Pruebas

El desarrollo de software requiere de un conjunto de actividades donde las posibilidades que aparezcan fallos de implementación o usabilidad son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, por lo que el desarrollo de software debe estar basado en garantizar la calidad desde el inicio. Las pruebas de software

constituyen un elemento importante para medir el grado en que se cumplen los requisitos del cliente.

Existen cuatro niveles de pruebas:

- **Pruebas unitarias:** se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño, ya sea un componente o un módulo del software [22].
- **Pruebas de integración:** es una técnica sistemática para confeccionar la arquitectura del software mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar componentes a los que se aplicó una prueba de unidad y construir una estructura de programa que determine el diseño [22].
- **Pruebas de sistema:** abarca una serie de pruebas diferentes cuyo propósito principal es ejercitar profundamente el sistema de cómputo. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se hayan integrado adecuadamente todos los elementos del sistema y que realizan las funciones adecuadas [22].
- **Pruebas de aceptación:** una vez culminado el proceso de pruebas por parte del equipo de desarrollo, es indispensable que el cliente verifique que el producto ha sido desarrollado con las normas y criterios establecidos, y cumple con todos los requisitos especificados por el cliente [23].

Para la aplicación de estos niveles existen los métodos de caja negra y caja blanca. En el capítulo 3 de la presente investigación se describen los métodos utilizados para validar la solución obtenida.

1.10.1 Método de caja negra

El método de caja negra se lleva a cabo sobre la interfaz del software, utilizando casos de pruebas durante el desarrollo de la actividad. Este método examina algunos aspectos del funcionamiento del software, sin tener en cuenta la estructura interna de este [24].

Estas pruebas permiten encontrar [24]:

1. Funciones incorrectas o ausentes
2. Errores de interfaz
3. Errores en estructuras de datos o en accesos a las bases de datos externas

4. Errores de rendimiento
5. Errores de inicialización y terminación

Existen varias técnicas para desarrollar el método de caja negra:

Técnica de la Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software [24].

Técnica del Análisis de Valores Límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables [24].

Técnica de Grafos de Causa - Efecto: permite al encargado de la prueba, validar complejos conjuntos de acciones y condiciones [24].

1.10.2 Método de caja blanca

El método de caja blanca se basa en un minucioso examen de los procedimientos. Se comprueban los caminos lógicos del software por casos de prueba que ejerciten conjuntos específicos de condiciones y se puede examinar el estado del programa en varios puntos, para determinar si el estado real coincide con el esperado o mencionado [24].

Las pruebas de caja blanca intentan garantizar que [24]:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo
- Se utilizan las decisiones en su parte verdadera y en su parte falsa
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas

Para ejecutar caja blanca se utiliza la técnica de camino básico**.** La misma permite al diseñador de casos de prueba, obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución [22].

1.11 Conclusiones parciales

- El análisis de los principios de la seguridad informática permitió identificar la prioridad de las políticas a controlar con la solución propuesta.

- El estudio de soluciones para el desarrollo de auditoría de seguridad informática permitió definir un conjunto de características para ser incorporadas a la solución propuesta.

- El estudio de metodologías y tecnologías para el desarrollo de software, permitió seleccionar las más adecuadas para la implementación de la solución propuesta.

CAPÍTULO 2: ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA.

2.1 Introducción

En el presente capítulo se explican las diferentes características que presenta el sistema propuesto. Se definen los principales elementos a tener en cuenta para la realización del mismo, tal es el caso de los requisitos funcionales y no funcionales, historias de usuario, diseño de la arquitectura, empleo de patrones de diseño y el modelo de datos.

2.2 Requisitos del software

Los requisitos de un sistema según el autor Ian Sommerville son la descripción de los servicios proporcionados por el mismo y sus restricciones operativas [25]. Para un mejor entendimiento de las necesidades del cliente y del sistema en cuestión, fue necesario definir un conjunto de requisitos funcionales y no funcionales.

2.2.1 Requisitos funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer en cuanto a funcionamiento, entradas, salidas y excepciones [25].
A través de las técnicas de tormentas de ideas, entrevistas con el cliente y la observación del comportamiento de las soluciones existentes para el desarrollo de auditorías de SI, se obtuvieron un total de 14 RF para el desarrollo del sistema, los cuales se listan en la tabla 5:

Tabla 5. Requisitos funcionales.

Nº	Nombre	Descripción	Complejidad	Prioridad para el cliente
RF1	Crear aplicación cliente.	Permite crear una aplicación para ser instalada en cada PC.	Media	Alta
RF2	Comprobar automáticamente que la PC esté en el dominio uci.cu.	Permite comprobar de forma automática que la PC esté en el dominio “uci.cu”.	Media	Alta

RF3	Comprobar automáticamente que el antivirus Kaspersky esté instalado en cada PC.	Permite comprobar que el antivirus Kaspersky esté instalado y actualizado en las PC.	Media	Alta
RF4	Comprobar automáticamente que el firewall (cortafuego) esté activado en cada PC.	Permite comprobar que el firewall esté activado en cada PC.	Media	Alta
RF5	Comprobar automáticamente que no existan carpetas compartidas en cada PC.	Permite comprobar que no existan carpetas compartidas en la PC.	Media	Alta
RF6	Comprobar automáticamente que el Sistema Operativo esté actualizado en cada PC.	Permite comprobar que el SO de la máquina esté actualizado.	Media	Alta
RF7	Planificar la ejecución automática de las comprobaciones de cada PC.	Permite realizar una planificación en cuanto al período de ejecución automática de las comprobaciones de cada una de las PSI en las PC.	Media	Media
RF8	Adicionar local	Permite adicionar un nuevo local.	Media	Media
RF9	Modificar local	Permite modificar un local.	Media	Media
RF10	Adicionar política	Permite adicionar una nueva política.	Media	Media
RF11	Modificar política	Permite modificar una política.	Media	Media
RF12	Buscar resultado de auditoría.	Permite hacer una búsqueda teniendo en cuenta los criterios: local, usuario, fecha, medio básico y políticas.	Media	Media
RF13	Enviar a un servidor el resultado de las comprobaciones de cada PC.	Permite enviar el resultado de cada una de las comprobaciones de las PSI en las PC a un servidor para ser almacenadas en este.	Media	Alta

RF14	Obtener reporte por local con el resultado de las comprobaciones de cada PC.	Permite obtener reportes desde un servidor, según criterios predefinidos, que muestren el resultado de las comprobaciones de cada una de las PSI realizadas en las PC.	Media	Alta

2.2.2 Requisitos no funcionales

Son restricciones de los servicios del sistema, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes y restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema [25]. Para el desarrollo de la presente investigación se definieron un total de 30 RNF, los cuales se clasifican en RNF de usabilidad, confiabilidad, soporte, restricciones del diseño, interfaz, de licencia, estándares aplicables y seguridad. A continuación, se expresan los requisitos no funcionales definidos para un mejor desarrollo del sistema:

- **Requisitos de usabilidad**
  - **Tipo de usuario final:**

RNF1. El sistema será usado por los responsables de velar por la Seguridad Informática en CEGEL, durante la realización de auditorías de SI en cada una de sus áreas o locales.

RNF2. El sistema estará administrado por:

Nombre y Apellidos	Nivel escolar	Cargo que ocupa	Años de experiencia	Experiencia con CEGEL
Willian Gómez Díaz	Universitario	Asesor de Seguridad Informática	3 meses	2 años

  - **Tipo de aplicación Informática:**

RNF3. El producto constituye una aplicación de escritorio enfocada en las necesidades de CEGEL para la realización de auditorías de Seguridad Informática tanto para Linux como para algunas estaciones de Windows utilizadas en el Centro.

- **Finalidad**

RNF4. Este sistema está enfocado en la realización de auditorías de Seguridad Informática en CEGEL.

- **Ambiente**

RNF5. El sistema debe presentar una interfaz legible y simple de usar.

RNF6. El tiempo de respuesta brindado por el sistema será menor de 5 segundos.

RNF7. El sistema se desarrolla con tecnología Java en su versión 1.8.60. Se utiliza además Glassfish Server en su versión 4.1.

RNF8. Se emplea como Sistema Gestor de Base de Datos PostgreSQL en su versión 9.4.1.

RNF9. Se emplea como marco de trabajo XEGFORT en su versión 2.0.

RNF10. Las PC clientes deben tener la máquina virtual de Java (JDK).

RNF11. El sistema se desarrolla en la plataforma Linux en la distribución Ubuntu 16.04.

RNF12. Las computadoras de los clientes no requerirán de muchas prestaciones.

RNF13. El idioma de todas las interfaces del sistema es el español.

RNF14. En el sistema no existen más de 3 interfaces para lograr una funcionalidad completa.

- **Requisitos de confiabilidad**

RNF15. El sistema debe estar disponible al menos durante toda la jornada laboral.

RNF16. El sistema presenta campos obligatorios para garantizar la integridad de la información que se introduce por el usuario.

RNF17. El sistema no permite la entrada de datos incorrectos.

RNF18. Ante el fallo de una funcionalidad del sistema, el resto de las funcionalidades que no dependen de esta deberán seguir funcionando.

- **Soporte**

RNF19. El uso del sistema no requiere un tiempo de preparación previa por los usuarios finales para su explotación debido a que es de fácil uso.

RNF20. Se debe recibir entrenamiento para la configuración y el mantenimiento del sistema.

- **Restricciones de diseño**

RNF21. Para el montaje del sistema se requiere del Sistema Gestor de Bases de Datos PostgreSQL en su versión 9.4.1.

RNF22. Las etiquetas de cada funcionalidad y los campos de cada interfaz deben tener títulos asociados a su función.

RNF23. El sistema presenta los términos capitalizados, es decir, la primera palabra tendrá su primera letra en mayúscula.

- **Interfaz**

RNF24. El sistema presenta una interfaz legible, simple de usar e interactiva.

RNF25. El sistema debe ser independiente de la plataforma.

RNF26. La tipografía y colores deben ser estándares en todo el sistema.

- **Requisitos de licencia**

RNF27. Para el desarrollo del sistema se emplean las siguientes licencias: PostgreSQL Licencie GNU GPL para el Gestor de Base de Datos PostgreSQL.

- **Estándares aplicables**

RNF28. Se emplea el estilo de codificación *CamelCase*.

- **Seguridad**

RNF29. El sistema brinda la posibilidad de establecer permisos sobre acciones, garantizando que solo acceda quien esté autorizado.

RNF30. El sistema muestra las funcionalidades de acuerdo a quien esté autenticado en el mismo.

RNF31. El sistema debe garantizar la protección ante acciones no autorizadas.

2.3 Historias de Usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [26]. En el desarrollo del sistema propuesto se confeccionaron un total de 14 historias de usuarios. A continuación, en la tabla 6 se muestra la historia de usuario Crear aplicación cliente.

Tabla 6. Historia de Usuario: Crear aplicación cliente.

Historia de usuario	
Número: 1	**Nombre del requisito:** Crear aplicación cliente.
Programador: Verónica Hernández Urgelles	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 40 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 40 horas
Descripción: Permite crear una aplicación capaz de comprobar las PSI en las PC y enviar los resultados a un servidor. Esta aplicación presenta una interfaz denominada “Formulario de instalación”, visible	

solamente en el momento en que se instale la aplicación. En la cual el usuario tiene que introducir la información que se necesita conocer (Local, Medio Básico, Usuario).

-En el campo Local se selecciona uno de los locales que aparecen.

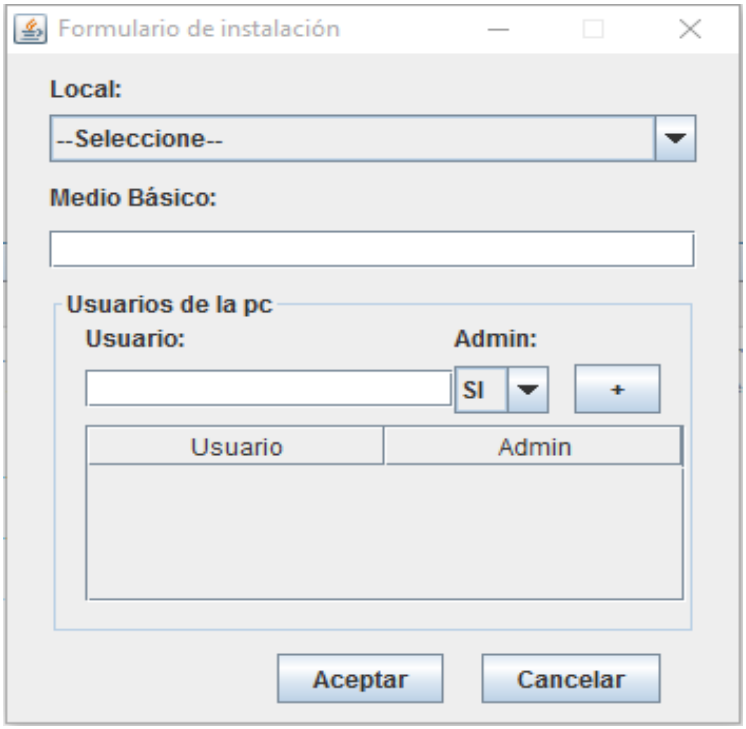
-El campo Medio Básico solo admite las letras “MB” en mayúscula seguidas de números de 8 a 15 dígitos.

-El campo Usuario de la PC solo admite cadenas de letras de 4 a 15 caracteres. Este campo posee un botón que lleva por nombre “+”, este se encarga de agregar en una tabla los usuarios que posee cada máquina, estos pueden ser administradores o no. La tabla con la información de los usuarios se encuentra ubicada debajo del campo Usuario.

Todos los campos tienen que ser completados, una vez que esto ocurra, se accede a la instalación de la aplicación mediante el botón “Aceptar”.

Observaciones: N/A

Prototipo elemental de interfaz gráfica de usuario:



2.4 Estimación de esfuerzo por Historia de Usuario

La tabla 7 contiene la estimación de esfuerzo por HU según el orden definido para la realización de cada una de estas. Esta estimación incluye todo el esfuerzo asociado a la implementación de las HU, la misma se expresa utilizando como medida el punto (máximo esfuerzo). Un punto se considera una semana ideal de trabajo sin ningún tipo de interrupción.

Tabla 7. Estimación de esfuerzo por HU.

Historia de Usuario	Puntos estimados
Crear aplicación cliente.	1
Comprobar automáticamente que la PC esté en el dominio uci.cu.	1
Comprobar automáticamente que el antivirus Kaspersky esté instalado en cada PC.	1
Comprobar automáticamente que el firewall (cortafuego) esté activado en cada PC.	1
Comprobar automáticamente que no existan carpetas compartidas en cada PC.	1
Comprobar automáticamente que el Sistema Operativo esté actualizado en cada PC.	1
Planificar la ejecución automática de las comprobaciones de cada PC.	1
Adicionar local	1
Modificar local	1
Adicionar política	1
Modificar política	1
Buscar resultado de auditoría.	1
Enviar a un servidor el resultado de las comprobaciones de cada PC.	1

Obtener reporte por local con el resultado de las comprobaciones de cada PC.	1

2.5 Plan de iteraciones

Una vez identificadas las HU y la estimación del esfuerzo por cada una de ellas, se realiza 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 durará cada una de estas. Teniendo en cuenta el riesgo para el desarrollo de cada una de las HU, el tamaño del equipo de desarrollo y que el número de HU no es grande, se decidió dividir el proyecto en 2 iteraciones, las cuales son detalladas a continuación:

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

Iteración 2: En esta iteración se implementarán las HU de prioridad media, las cuales son tan importantes como las mencionadas anteriormente, pero al depender de la implementación de estas, se determinó realizarlas en otra iteración.

Tabla 8. Plan de iteraciones.

Iteraciones	Orden de las HU por iteración	Duración de las iteraciones por semana
Iteración 1	Crear aplicación cliente.	4
	Comprobar automáticamente que la PC esté en el dominio uci.cu.	
	Comprobar automáticamente que el antivirus Kaspersky esté instalado en cada PC.	
	Comprobar automáticamente que el firewall (cortafuego) esté activado en cada PC.	
	Comprobar automáticamente que no existan carpetas compartidas en cada PC.	

Iteración 2	Comprobar automáticamente que el Sistema Operativo esté actualizado en cada PC. Planificar la ejecución automática de las comprobaciones de cada PC.	4
	Adicionar local	
	Modificar local	
	Adicionar política	
	Modificar política	
	Buscar resultado de auditoría.	
	Enviar a un servidor el resultado de las comprobaciones de cada PC. Obtener reporte por local con el resultado de las comprobaciones de cada PC.	

2.6 Plan de entrega

El plan de entrega se planifica mediante la realización de reuniones con el equipo de desarrollo y el cliente, donde se definen el tiempo de realización del sistema informático. Teniendo en cuenta la duración de cada iteración, se planifican las entregas. La tabla 9 muestra la planificación de entregas para la solución propuesta.

Tabla 9. Plan de entregas.

Herramienta	Final de la iteración 1 1ra semana de mayo	Final de la iteración 2 1ra semana de junio
Sistema Informático para la realización de auditorías de SI	0.5	1.0

2.7 Tarjetas CRC (Clase, Responsabilidad, Colaborador)

La principal tarea de las tarjetas CRC es propiciar un enfoque orientado a objetos. 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 clases con las que interactúa. A continuación, en la tabla 10 se detalla la tarjeta CRC de la clase *ComprobarPoliticas*.

Tabla 10. Tarjeta CRC: *ComprobarPoliticas*.

ComprobarPoliticas	
Responsabilidad	**Colaborador**
Permite chequear las políticas de Seguridad mediante el funcionamiento de comandos, tanto para linux como para Windows, que se pueden ejecutar por consola o terminal.	*java.net.InetAddress* *java.util.Properties* *java.io.BufferedReader* *Runtime.getRuntime()*

2.8 Tareas de ingeniería

Las tareas de ingeniería permiten a los desarrolladores obtener un nivel de detalle más avanzado de las HU. A continuación, en la tabla 11 se describe la tarea de ingeniería 1, que responde a la HU: Crear aplicación cliente.

Tabla 11. Tarea de de ingeniería: Implementar HU-Crear aplicación cliente.

Tarea de ingeniería	
Número de tarea: 1	**Historia de Usuario (No.1):** Crear aplicación cliente
Nombre de la tarea: Implementar HU-Crear aplicación cliente.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1 y 1/2 semana
Fecha de inicio: 28-4-2016	**Fecha de fin:** 8-4-2016
Programador responsable: Verónica Hernández Urgelles	

Descripción: Mediante este requisito se debe obtener una aplicación capaz de comprobar las políticas de Seguridad informática en cada PC.

2.9 Arquitectura del sistema

La Arquitectura de Software (AS) es la organización fundamental de un sistema, encargada de sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución [27]. Para el sistema propuesto se utilizó una arquitectura Cliente-Servidor aplicando el patrón arquitectónico N-capas, específicamente se definieron 4 niveles o capas, las cuales se describen a continuación.

En el Cliente se encuentran la Capa de Presentación, que contiene las vistas de la aplicación y las acciones, y la Capa de Negocio del Cliente, la cual contiene los gestores de negocio del cliente. Las acciones se encargan de manejar la información existente en los formularios, permitiendo que los gestores de negocio del cliente realicen las validaciones necesarias. En el Servidor se encuentra la Capa de Negocio del Servidor, la Capa de Acceso a Datos y las Entidades Persistentes. La primera contiene los gestores de negocio del servidor que se encargan de realizar validaciones más cercanas al negocio delegando en los gestores de acceso a datos la persistencia de la información obtenida, a través de las entidades persistentes. Transversal a ellas se encuentran las Entidades del Dominio, responsables de transportar la información desde el Servidor hasta el Cliente. En la figura 1 se muestra la estructura de las capas antes descritas.

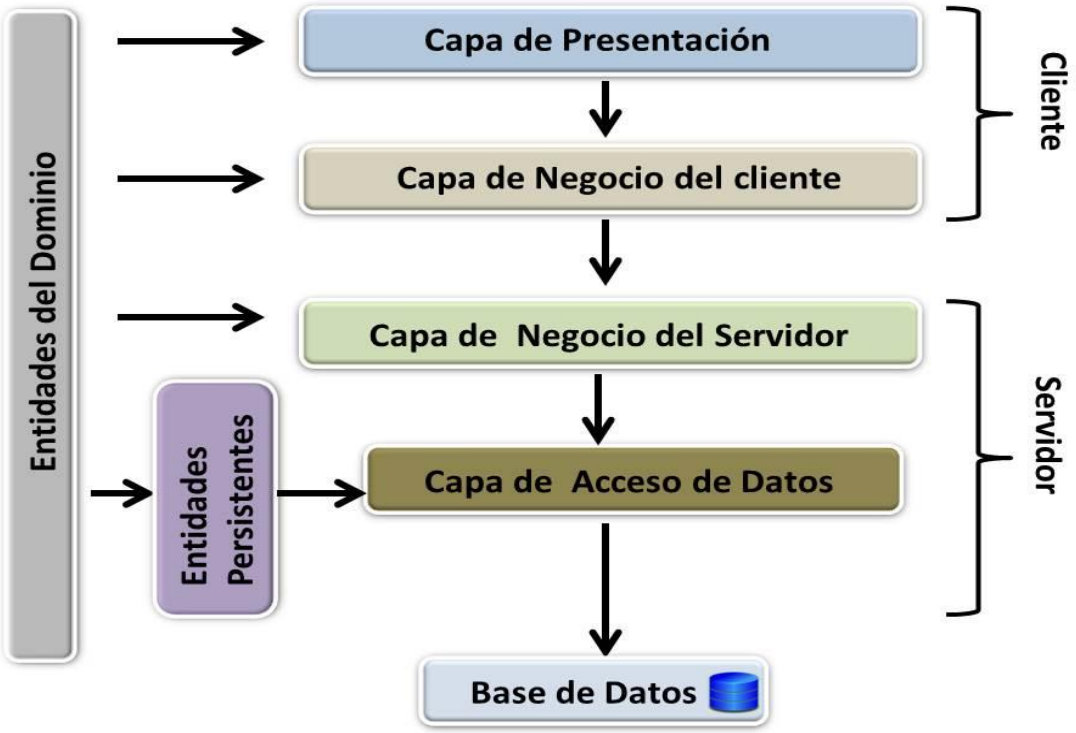


Figura 1. Modelo N-capa definido para el sistema propuesto (Fuente: elaboración propia).

2.10 Diagrama de paquetes del sistema

Los diagramas de paquetes tienen el objetivo de obtener una visión clara del software en desarrollo, creando una organización de los subsistemas que lo componen, agrupando los elementos del análisis y diseño y detallando las relaciones de dependencia entre ellos. El mecanismo de agrupación utilizado en este tipo de diagrama se denomina Paquete [30]. En la figura 2 se representa el diagrama de paquetes utilizado para organizar los subsistemas que componen la solución propuesta.

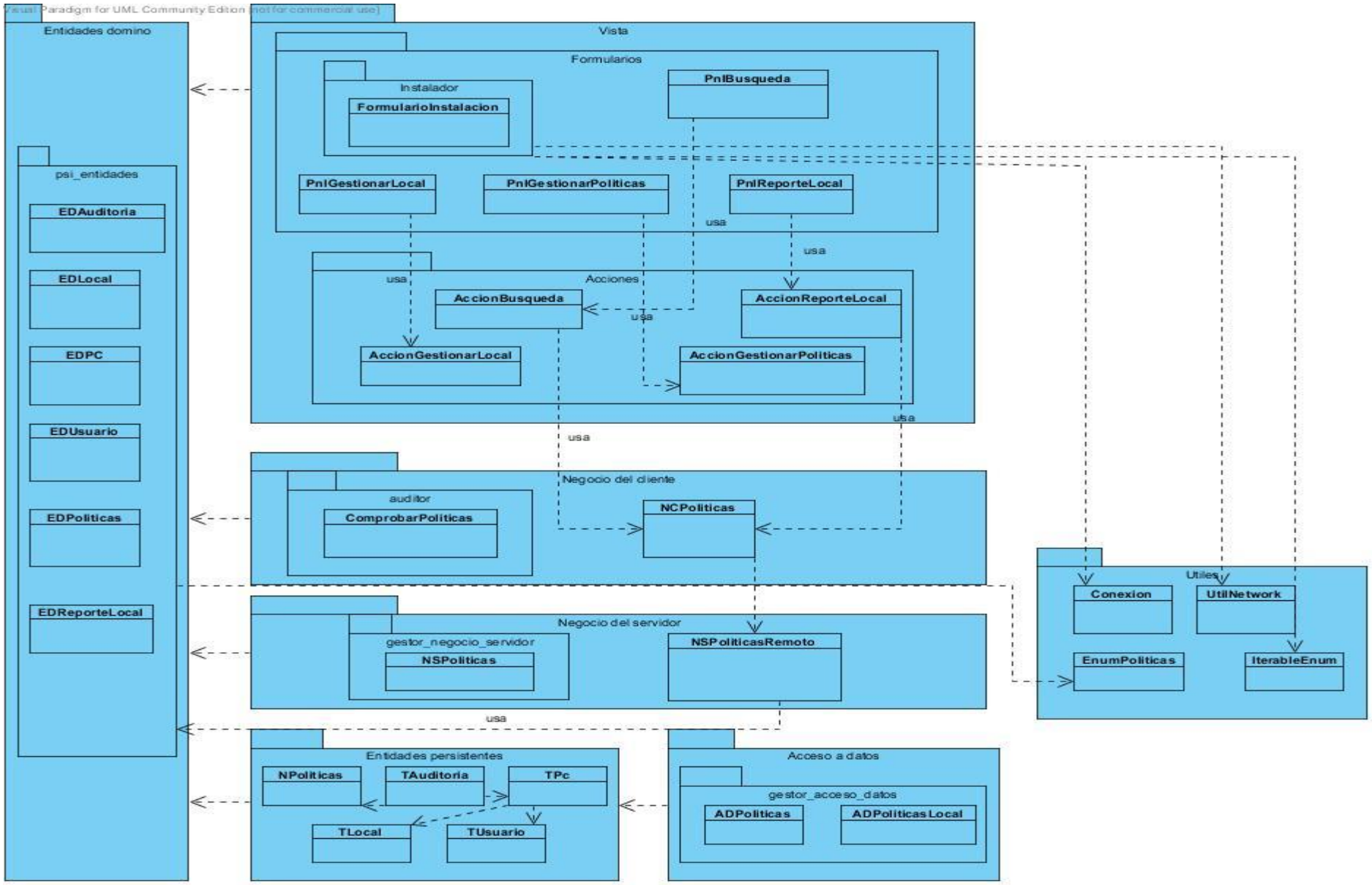


Figura 2. Diagrama de paquetes del sistema propuesto (Fuente: elaboración propia).

El paquete *Vista* está compuesto por los paquetes *Formularios* y *Acciones*. El paquete *Formularios* contiene al paquete *Instalador* (compuesto por la interfaz de instalación del cliente) y las interfaces que le permitirá al usuario interactuar con el sistema. El paquete *Acciones* contiene las clases que controlan la interfaz de usuario. El paquete de *Negocio del cliente* está compuesto por el paquete *Auditor* (contiene la clase encargada de comprobar las políticas en la PC) y la clase *NCPoliticas*. El paquete *Negocio del servidor* contiene al paquete *gestor_Negocio _Servidor* (compuesto por la clase *NSPoliticas*) y la clase *NSPoliticasRemoto*. El paquete *Entidades persistentes* contiene las clases necesarias para el acceso a la base de datos y de realizar las consultas en la misma. El paquete *Acceso a datos* que contiene a su vez el paquete *gestor_acceso_datos* compuesto por las clases *ADPoliticas* y *ADPoliticaLocal*. Paralelo a estos paquetes se encuentra el paquete *Entidades del domino,* el cual contiene las clases controladoras encargadas de mapear la base de datos, este paquete se relaciona directamente con los paquetes principales antes mencionados. También se encuentra el paquete Útiles que contiene las funcionalidades comunes para todos los paquetes, tales como las clases *IterableEnumeration*, *Conexion*, *EnumPoliticas* y la clase *UtilNetwork* que es la encargada de mostrar información de la PC.

2.11 Modelo de datos

El modelo de datos describe la estructura y elementos de la base de datos y la forma en que se relacionan entre sí. En la figura 3 se muestra el modelo de datos del sistema propuesto. Este está compuesto por 5 tablas: la *tabla t_pc* que posee una relación de uno a mucho con la tabla *t_auditoría* y con la tabla t_usuario, la tabla *t_local* tiene una relación uno a mucho con la *tabla t_pc* y la *tabla n_políticas* está representada por un nomenclador debido a que sus datos persisten en el tiempo, esta tabla tiene una relación de uno a mucho con la tabla *t_auditoría*.

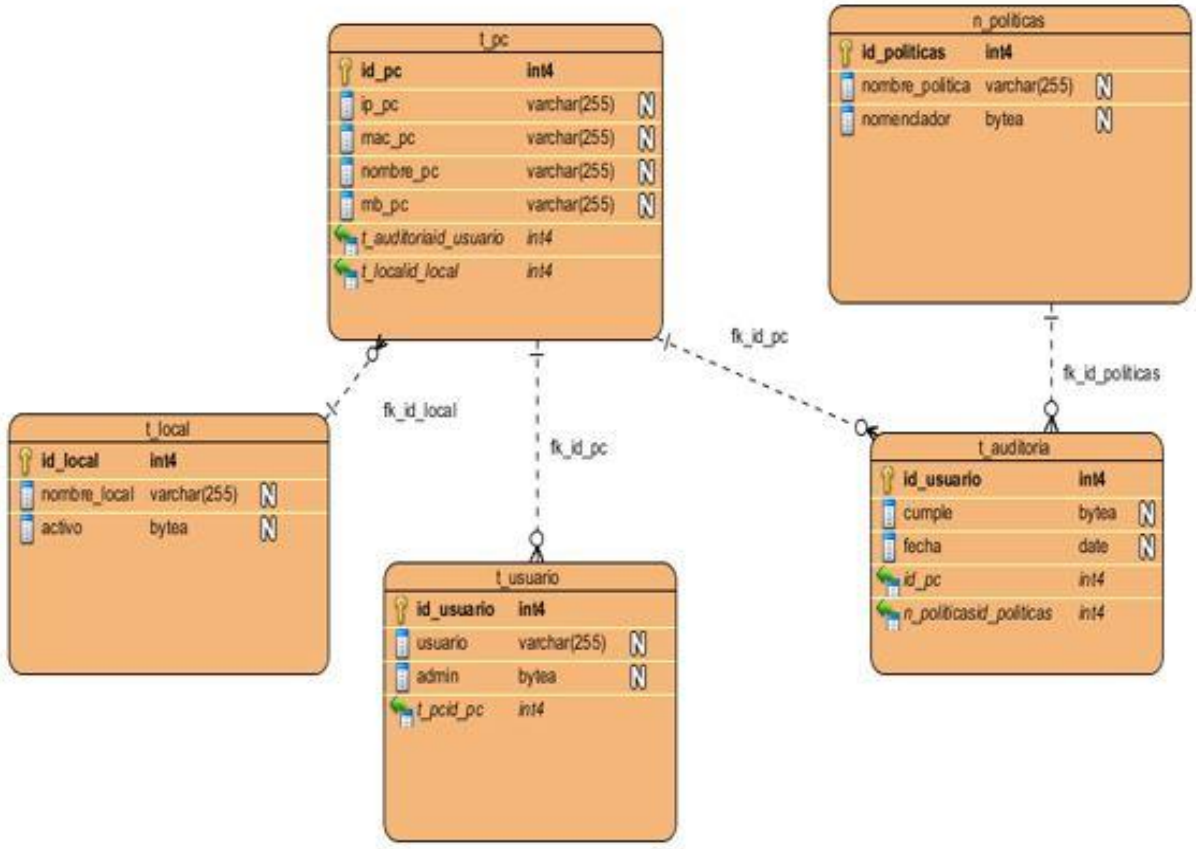


Figura 3. Modelo de datos del sistema propuesto (Fuente: elaboración propia).

2.12 Patrones de diseño

En el diseño de las clases del sistema propuesto se aplicaron patrones del grupo GRASP y GoF. A continuación, en los próximos epígrafes se describen los patrones utilizados.

2.12.1 Patrones GRASP

Los patrones GRASP, constituyen buenas prácticas en el diseño de software. Estos guardan relación directa con la creación y asignación de responsabilidades a los objetos [28]. Para el diseño del sistema se emplearon los siguientes patrones GRASP:

Experto: Este patrón es utilizado en las clases entidades ya que estas contienen toda la información correspondiente a las tablas de la base de datos y se encargan de representar las columnas como atributos y métodos, todo esto a nivel de aplicación.

Creador: Este patrón se utiliza en las clases controladoras, pues las mismas se encargan de crear objetos de las entidades**.**

Controlador: En el sistema el uso de este patrón se evidencia en las clases controladoras, ejemplo las clases *AppInstalador*, *AppAuditor,* encargadas de controlar el negocio.

2.12.2 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, pues la comunicación se realiza a través de una única interfaz (la clase FachadaGestoresRemoto es el único punto de acceso mediante el cual el cliente se conecta con el servidor).

2.13 Estándar de codificación

Para una mejor estandarización en el código del sistema se emplea el estándar de codificación CamelCase en sus variantes *lowerCamelCase* y *upperCamelCase*, lo que hace más sencillo el análisis del código y el mantenimiento del sistema.

Convenciones de nomenclatura

General:

- En caso de existir palabras, con la letra ñ, se empleará el término nn.
- Para las variables, clases o métodos que contengan tildes serán utilizadas las propias palabras, pero sin la acentuación.
- Serán utilizados nombres relacionados con el contexto en que se desarrolla.

Identación:

El código del sistema será identado por K&R de 4 espacios.

Clases:

- El nombre de las clases siempre comenzará con mayúscula. En caso de ser una palabra compuesta, cada una de las palabras comenzarán también de la misma forma.

- Serán utilizados los comentarios en las clases y en las funciones según sea necesario, así como las líneas y espacios en blanco entre funciones y dentro de las mismas llaves.

Nombre de variables:

- Los nombres de las variables no serán ambiguos.

El correcto uso de cada una de estas nomenclaturas es de gran importancia para el desarrollo del sistema informático debido a que sirven de guía para el entendimiento y estandarización del código del sistema, facilitando su comprensión para la realización de mantenimientos posteriores.

2.14 Descripción del sistema

El sistema obtenido está compuesto por dos subsistemas. El primero se ejecuta de forma automática en períodos de tiempos planificados en cada una de las PC y envía el resultado de las comprobaciones hacía un servidor, en el cual se almacena el historial de cada una de las auditorías realizadas a las máquinas. Por otra parte, el segundo subsistema permite por medio de una interfaz consultar el servidor y obtener a través de reportes el resultado de las auditorías, organizados por locales o períodos de tiempo. A continuación, se muestra en la figura 4 la estructura del sistema.

Figura 4. Estructura del sistema para la realización de auditorías de SI (Fuente: elaboración propia).

El sistema contará con dos niveles de usuario:

- **Administrador** (Asesor de SI): posee control total del sistema, por lo que puede realizar cualquier acción sobre los módulos que presenta el mismo.
- **Usuario básico** (Activistas por laboratorios, Director, Sub-director, Jefes de Departamentos)**:** pueden obtener cada uno de los reportes con los resultados de las auditorías por locales y períodos de tiempo.

2.15 Conclusiones parciales

- El empleo de la metodología XP en función de describir el desarrollo de la solución propuesta, permitió realizar un trabajo bien estructurado, generando los artefactos necesarios en cada fase, en correspondencia con el cronograma de desarrollo propuesto.

- El uso de una arquitectura N-capas y el empleo de patrones de diseño, garantizan obtener una solución de software con poca dependencia entre clases, flexible al mantenimiento y a la aceptación de cambios.

- La estructuración de la solución propuesta en dos subsistemas, constituye la base para la obtención de reportes con el estado de las políticas de seguridad informática en las PC del Centro de Gobierno Electrónico.

CAPÍTULO 3: VALIDACIÓN DEL SISTEMA

3.1. Introducción

En el presente capítulo se describe la fase producción de la metodología de desarrollo empleada, en la cual se le realizaron chequeos y pruebas al sistema desarrollado antes ser entregada al cliente. Además, se exponen los artefactos que se obtuvieron en esta fase, tales como: el acta de liberación y el acta de aceptación.

3.2. Técnicas de validación de los requisitos

La aplicación de técnicas de validación de requisitos tiene como objetivo demostrar que los requisitos previamente definidos están en correspondencia con los solicitados por el cliente. En este caso, para el desarrollo del sistema, se emplearon las siguientes técnicas:

- **Revisiones formales de los requisitos**: se realizaron revisiones formales a cada uno de los requisitos por parte del cliente y del equipo de desarrollo, obteniéndose no conformidades de tipo técnicas y de ortografía, las que fueron corregidas satisfactoriamente en tiempo. Con el cliente se llevaron a cabo 3 revisiones. En la primera, 5 de los requisitos funcionales fueron modificados con el propósito de mejorar la redacción de los mismos. En el segundo encuentro fueron modificados 2 requisitos por la misma causa. En un tercer encuentro, el cliente quedó satisfecho con el trabajo efectuado por el equipo de desarrollo y se aprobaron finalmente la totalidad de los requisitos, generándose de este encuentro el Acta de Aceptación de los requisitos (ver Anexo 1).

- **Generación de casos de prueba**: como parte del proceso de validación de los requisitos funcionales del sistema, se diseñaron un total de 6 casos de pruebas a partir de las historias de usuario definidas. Estos artefactos se generaron con el propósito de verificar que la descripción de los requisitos identificados, está libre de ambigüedades y muestra la información necesaria para ser comprobados a través de la realización de pruebas funcionales al concluir la implementación del sistema.

3.3. Validación del diseño

Para comprobar la calidad del diseño se emplearon las métricas de diseño Relaciones entre clases (RC) y Tamaño operacional de clase (TOC).

3.3.1 Relaciones entre clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra. Permite evaluar el acoplamiento, la complejidad de mantenimiento, la reutilización y la cantidad de pruebas de unidad necesarias para probar una clase, teniendo en cuenta las relaciones existentes entre ellas [29]. Para determinar el grado de afectación de los atributos de calidad que mide la métrica RC es necesario determinar la Cantidad de Relaciones de Uso (CRU) que posee cada una de las clases a medir. Una vez determinada la CRU, se procede a calcular el promedio de las mismas y teniendo ambos valores según los criterios expuestos en el Capítulo 1, se determina la incidencia de los atributos de calidad en cada una de las clases. La aplicación del instrumento de evaluación de la métrica RC en el diseño del sistema propuesto, arrojó los resultados expuestos en el gráfico de la figura 5.

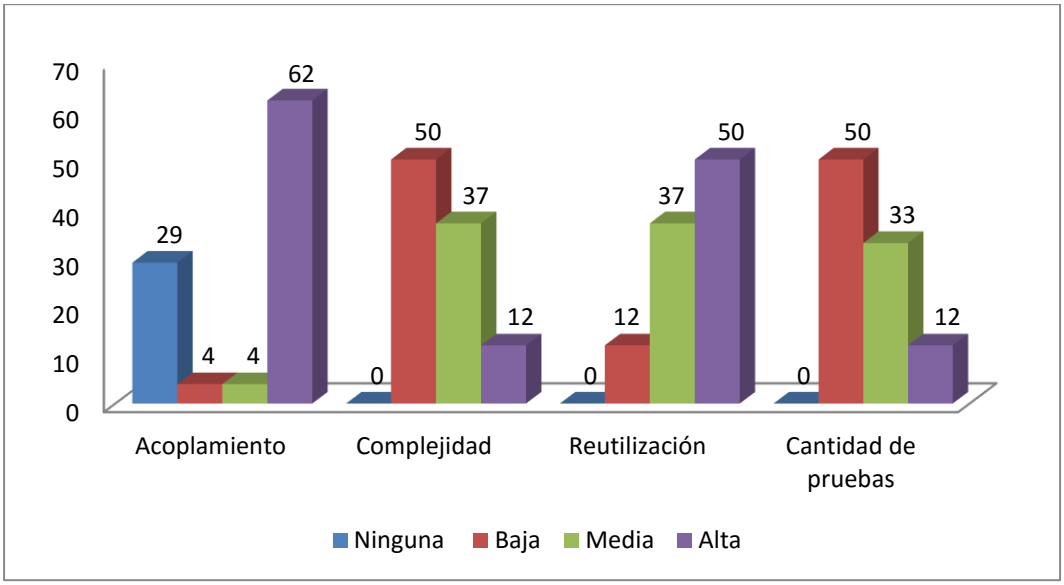


Figura 5. Resultado de la métrica RC (Fuente: elaboración propia).

- **Acoplamiento**: según los resultados que se muestran, el 62% de las clases posee alto acoplamiento.

- **Complejidad de mantenimiento:** según los resultados que se muestran en la figura, el 50% de las clases se comportan de forma satisfactoria pues, son de fácil soporte.
- **Reutilización:** según los resultados que se mostraron en la figura, el 50% de las clases tiene un alto grado de reutilización.
- **Cantidad de pruebas:** luego de aplicar la métrica se obtuvo que el 50% de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones y pruebas de software.

Según lo analizado anteriormente, los valores de RC se comportan de forma satisfactoria, en cada una de los resultados de interés para el diseño propuesto. Estos expresan que las clases del diseño presentan alto acoplamiento, parámetro que no afecta el diseño, teniendo en cuenta las características del negocio. Por otra parte, se muestra que la complejidad de mantenimiento y la cantidad de pruebas son bajas, en consecuencia, el grado de reutilización es alto.

3.3.2 Tamaño Operacional de las Clases (TOC)

Para determinar el valor de los atributos de calidad, se debe determinar la cantidad de procedimientos que posee cada una de las clases a medir. Una vez determinado la CP se procede a calcular el promedio del mismo y, según los criterios expuestos en el Capítulo 1, se determina la incidencia de los atributos de calidad en cada una de las operaciones de las clases. La aplicación del instrumento de evaluación de la métrica TOC en el diseño del sistema propuesto, arrojó los resultados expuestos en el gráfico de la figura 6.

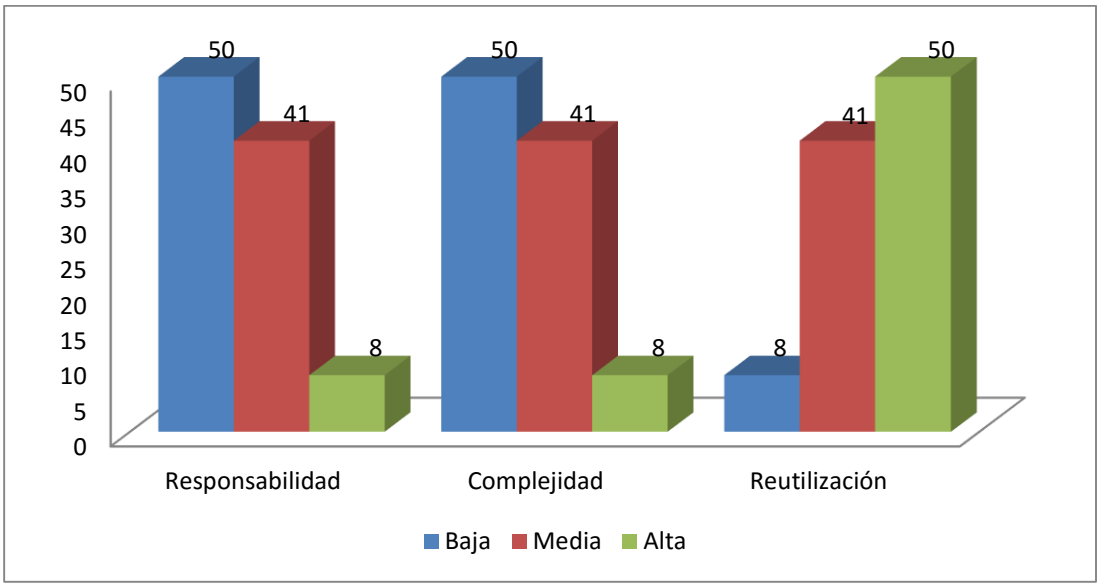


Figura 6. Resultado de la métrica TOC (Fuente: elaboración propia).

- **Responsabilidad:** luego de aplicar la métrica se obtuvieron resultados satisfactorios que reflejan una responsabilidad baja con un valor del 50%.
- **Complejidad de Implementación:** después de haberse realizado la medición de la métrica, arrojó también resultados positivos ya que la complejidad de las clases es baja en un 50 %.
- **Reutilización:** se obtuvieron valores que según muestra la gráfica de la figura anterior se comporta en un nivel alto con un 50%.

Los resultados arrojados por la métrica TOC expresan que las clases del diseño del sistema propuesto presentan una elevada reutilización y baja complejidad y responsabilidad. Por lo que se concluye que los resultados obtenidos con esta métrica son positivos.

3.4. Pruebas

Para garantizar un correcto funcionamiento del sistema, que permita cumplir con las expectativas del cliente, es necesario realizar pruebas. Este proceso constituye una de las principales fortalezas de la metodología XP, debido a que se realiza de manera continua, garantiza que los errores sean detectados en un corto plazo de tiempo y se corrijan de manera más sencilla, asegurando el éxito del software. Existen diferentes niveles y métodos de pruebas, descritos en el Capítulo 1 de la presente investigación. En los próximos epígrafes se describen los métodos y niveles utilizados en la validación de la solución propuesta.

3.4.1 Método de caja blanca

El método de caja blanca garantiza que se ejerciten por lo menos una vez todos los caminos independientes del código, así como la ejecución de todos los bucles en sus límites operacionales y todas las decisiones lógicas en las vertientes verdaderas y falsas [30]. Para su aplicación a las funcionalidades implementadas, se empleó la técnica del camino básico. A continuación, se demuestra el desarrollo de la técnica tomando como ejemplo el método *agregarUsuario()*, perteneciente al Formulario de instalación. La aplicación de esta técnica permitió obtener una medida de la complejidad lógica para el diseño de los casos de prueba (CP) y usar dicha medida como guía para la definición de un conjunto de caminos básicos de ejecución. En la figura 7 se muestra el método seleccionado.

```
private void agregarUsuario(java.awt.event.ActionEvent evt) {
    DefaultTableModel modelo = (DefaultTableModel) jtUsuarios.getModel();//1
    if (!txfUsuario.getText().isEmpty()) {//2
        if (validarUsuario()) {//3
            modelo.addRow(new Object[]{txfUsuario.getText(), jComboBox1.getSelectedItem().equals("SI")});//4
            txfUsuario.setText("");
            jtUsuarios.setModel(modelo);
        }else{
            JOptionPane.showMessageDialog(this, "Debe introducir un valor válido en el campo Usuario.","Información",JOptionPane.INFORMATION_MESSAGE);//5
        }
    } else {
        JOptionPane.showMessageDialog(this, "Debe introducir un valor en el campo Usuario.","Información",JOptionPane.INFORMATION_MESSAGE);//6
    }
   //7
}
```

Figura 7. Método *agregarUsuario()* (Fuente: elaboración propia).

A continuación, se describen los pasos que se realizaron para desarrollar la técnica del camino básico:

1. **Confeccionar el grafo de flujo:** usando el código de la figura 7 se realizó la representación del grafo de flujo, el cual está compuesto por los siguientes elementos:

- Nodos: son círculos que representan una o más sentencias procedimentales.
- Aristas: son flechas que representan el flujo de control y son análogas a las flechas del diagrama de flujo.

- Regiones: son las áreas delimitadas por aristas y nodos.

A continuación, la figura 8 muestra el grafo de flujo obtenido:

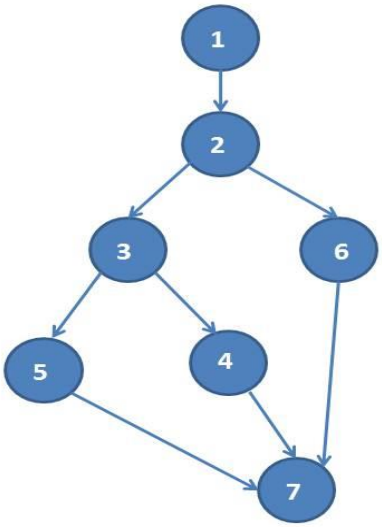


Figura 8. Grafo del camino básico del método *agregarUsuario()* (Fuente: elaboración propia).

2. **Calcular la complejidad ciclomática:** proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado define el número de caminos independientes del conjunto básico de un programa, la complejidad ciclomática se calcula de tres formas distintas. En la solución propuesta se aplican las tres variantes con el propósito de triangular los resultados obtenidos, validando que la cantidad de caminos a definir es la correcta:

Variante 1:

V (G)= A-N+2, donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

V (G) = A-N+2 = 8-7+2 = 3

Variante 2:

Teniendo en cuenta el número de regiones que presenta el grafo de flujo obtenido, se definen un total de 3 regiones.

Variante 3:

V(G) = Nodos de predicado + 1, donde los nodos predicados son los que tienen como salida más de una arista.

V(G) = Nodos de predicado + 1

V(G) = 2 + 1 = 3

Las tres variantes aplicadas brindan como resultado que el valor de la complejidad ciclomática del método *agregarUsuario()* es igual a 3.

3. **Determinar un conjunto básico de caminos linealmente independientes:** una vez aplicadas las tres variantes para calcular la complejidad ciclomática del método *agregarUsuario(),* se obtiene que el número de caminos linealmente independientes de la estructura de control del método es 3, definiéndose los siguientes caminos:
   Camino básico #1: 1-2-3-4-7
   Camino básico #2: 1-2-3-5-7
   Camino básico #3: 1-2-6-7
4. **Determinar un conjunto básico de caminos linealmente independientes:**
   Obtención de casos de prueba: cada camino independiente es un caso de prueba a realizar, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino. En este caso se obtuvieron 3 caminos básicos, por tanto, se hace necesario la confección de igual número de CP, para aplicar las pruebas a este método. A continuación, en la tabla 12 se muestra el CP diseñado para el camino básico # 3:

Tabla 12. Caso de prueba de caja blanca para el camino básico #3

Caso de pruebas: Camino básico # 3	
Entrada	El usuario de la PC
Resultados esperados	Se muestra un mensaje indicando que se debe introducir un valor en el campo usuario
Condiciones	El campo usuario debe quedar vacío

Una vez ejecutados todos los casos de pruebas obtenidos a través de la aplicación de la técnica camino básico para todos los métodos del sistema, se concluye que el código generado está libre de ciclos infinitos y códigos innecesarios.

3.4.2 Método de caja negra

El método de caja negra se aplica con el propósito de comprobar las funcionales del software, a través de la técnica de partición equivalente y valores límites, utilizando CP, con el objetivo de encontrar errores de funciones incorrectas o ausentes, de interfaz o en accesos a la base de datos. Para la aplicación de este método sobre la solución desarrollada, se diseñaron un total de 6 CP, que constituyen una guía para comprobar el correcto funcionamiento del sistema obtenido. A continuación, en la tabla 13 se muestra el diseño del CP definido para la HU: Crear aplicación cliente.

Tabla 13. Caso de prueba de caja negra de la HU: Crear aplicación cliente

Escenario	Descripción	Local	MB	Usuario	Admin	Respuesta del sistema
EC 1.1 Datos correctos.	Instalar la aplicación cliente en las PC.	Lab-419	MB01952326	vhernandez	Sí	Las aplicación se instaló satisfactoriamente
		Lab-419	I (*95668)	vhernandez	Sí	Por favor revise la sintaxis del campo.
		Lab-419	MB01952326	I (*juh)	Sí	Por favor revise la sintaxis del campo.
EC 1.2: Datos incompletos.	Instalar la aplicación cliente en las PC.	I (Vacío)	MB01952326	vhernandez	Sí	Debe llenar todos los campos.
		Lab-419	I (Vacío)	vhernandez	Sí	Debe llenar todos los campos.
		Lab-419	MB01952326	I (Vacío)	Sí	Debe llenar todos los campos.

Con el objetivo de comprobar que las funcionalidades del sistema fueron implementadas correctamente y responden a las necesidades del cliente, aplicando los CP antes descritos, se realizaron pruebas funcionales por parte de los especialistas del grupo de Calidad del Centro de Gobierno Electrónico. Las pruebas se realizaron en dos iteraciones. En la primera se detectaron un total de 21 No Conformidades (NC), clasificadas en 6 de ortografía, 12 de redacción y 3 de validación, al finalizar la iteración todas las NC quedaron resueltas. En la segunda iteración los resultados fueron satisfactorios, obteniéndose cero NC, generándose así el Acta de Liberación Interna de Productos de Software (ver Anexo 3). La figura 9 ilustra los resultados de aplicar el método de caja negra, teniendo en cuenta los tipos de NC identificadas (ortografía, redacción y validación).

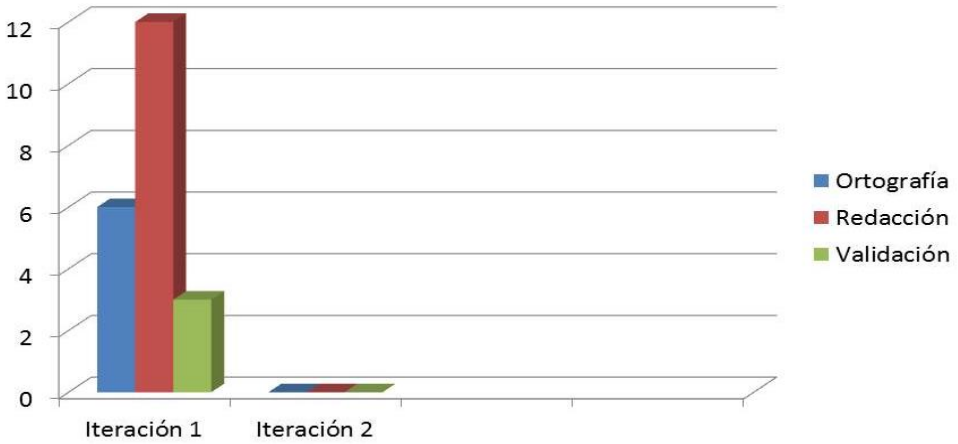


Figura 9. No conformidades detectadas al aplicar el método de caja negra (Fuente: elaboración propia).

3.4.3 Pruebas de aceptación

La prueba de aceptación es generalmente desarrollada y ejecutada por el cliente en conjunto con el equipo de desarrollo. Esta tiene como objetivo determinar cómo el sistema satisface sus criterios de aceptación, validando los requisitos identificados para el desarrollo del software, incluyendo la documentación y procesos de negocio. Está considerada como la fase final del proceso, para crear un producto confiable y apropiado para su uso [29].

Para la aplicación de estas pruebas a la solución obtenida, se confeccionaron casos de prueba de aceptación por cada HU. Seguidamente la tabla 14 muestra el caso de prueba de aceptación para la HU: Crear aplicación cliente.

Tabla 14. Caso de prueba de Aceptación de la HU: Crear aplicación cliente.

<table>
<tr><th colspan="3">Caso de prueba de aceptación</th></tr>
<tr><td colspan="2">Código de caso de prueba: 01</td><td>Nombre historia de usuario: Crear aplicación cliente.</td></tr>
<tr><td colspan="3">Nombre de la persona que realiza el caso de prueba: Verónica Hernández Urgelles</td></tr>
<tr><td colspan="3">Descripción de la prueba: comprobar que la aplicación cliente se instala de forma satisfactoria en las PC con cada uno de los datos definidos en la descripción de la HU.</td></tr>
<tr><td colspan="3">Condiciones de ejecución: Se debe tener instalado en las PC donde se vaya a instar la aplicación la máquina virtual de java (JDK).</td></tr>
<tr><td colspan="2">Entrada/Pasos de ejecución:
N/A</td><td rowspan="2">Resultados esperados:</td></tr>
<tr><td>Acción:</td><td>Entrada:</td></tr>
<tr><td>Se pulsa el botón aceptar del formulario para la instalación.</td><td>Texto.</td><td>Si se instala correctamente, el sistema lanza un mensaje de diálogo indicando que la aplicación se instaló satisfactoriamente.</td></tr>
<tr><td colspan="3">Evaluación de prueba: Satisfactoria</td></tr>
</table>

Se realizó un encuentro con el Ing. Yordanis García Leiva, especialista en temas de seguridad informática en CEGEL, con el objetivo de validar que el sistema obtenido cumple con las necesidades del cliente, teniendo en cuenta los CP definidos. Los resultados obtenidos fueron satisfactorios, avalados por el especialista entrevistado como una

herramienta de apoyo para el desarrollo de las auditorias de SI en el centro. Al finalizar el encuentro se generó el Acta de Aceptación del Producto (ver Anexo 2).

3.5. Conclusiones parciales

- Las técnicas de validación de requisitos y métricas de diseñó utilizadas, garantizaron iniciar un desarrollo libre de anomalías e incongruencias en el análisis y diseño del sistema.

- Los métodos de pruebas aplicados demostraron la obtención de un sistema libre de errores, que responde a las necesidades del cliente.

CONCLUSIONES GENERALES

- El estudio de las soluciones existes para el apoyo a la realización de auditoría de SI, permitió identificar las características comunes de estos softwares e incorporarlas al sistema desarrollado para la realización de auditoría de SI en CEGEL.

- La estructuración de la solución propuesta a partir de la utilización de patrones de diseño y el uso de una arquitectura n-capas, permitió obtener un sistema mantenible y escalable.

- La aplicación de técnicas de validación de requisitos, métricas de validación del diseño, pruebas unitarias y de aceptación a la solución propuesta, certifican la obtención de un software funcional que responde a los requisitos del cliente, avalado en cada caso por las actas de liberación y aceptación emitidas.

RECOMENDACIONES

- Ampliar el alcance del sistema de manera que permita controlar nuevas políticas de seguridad informática durante el desarrollo de auditorías.

- Incrementar los criterios de búsquedas con el propósito de obtener nuevos reportes que permitan a la dirección de CEGEL tener una información detallada sobre el cumplimiento de las políticas de seguridad informática en el área.

- Extender el uso del sistema a otras áreas de la universidad.

BIBLIOGRAFÍA REFERENCIADA

1. **Pfleeger, Charles P**. *Security in computing*. 2006. ISBN: 978-0-13-239077-4
2. **Labrador, Ramón m. Gómez**. E.t.s de ingenieria informática. Universidad de Sevilla. 2013. Disponible en: http://www.informática.us.es/~ramon/tesis/CORBA/Seminario-MASIF/
3. **Mirabal Sarria, Yamilet y Maragoto, María Isabel**. Propuesta de una guía de seguridad informática integrada a la gestión de la calidad. Pinar del Rio. 2011.
4. **Rivas, Gonzalo Alonso**. *Auditoría informática*. Ediciones Díaz de Santos, 1989.
5. **Benítez Moisés**, Gestión Integral. *Políticas de Seguridad informática*. 2013. Disponible en: http://www.gestionintegral.com.co/wp-content/uploads/2013/05/Políticas-de-Seguridad-Informática-2013-GI.pdf
6. **Nessus**. Disponible en: http://www.nessus.org/news-events/pr...bility-scanner
7. **Nmap security scanner**. 2016. Disponible en: https://nmap.org
8. **Matos, Yulissa Torres y González, Yanelis Morales**, "Plataforma de Gestión de Hardware y Software. Módulo: Mapa tecnológico". Tesis de Diploma, Universidad de las Ciencias Informáticas. 2011.
9. **A. Virrueta Méndez**, *Metodología de Desarrollo de Software*, Instituto Tecnológico superior de Apatzingán, Apatzingán Michoacán, México, 2010.
10. **Metodología Ágil: SCRUM**. Usado para el desarrollo de software, 2015. Disponible: http://apuntesusach.herokuapp.com/tutorial/scrum
11. **Sommerville, Ian**. 2004. *Software Engineering*. International computer science series. ed: Addison Wesley.
12. **Visual Paradigm Team**. 2014, All-in-One, End-to-End Information Technology System Modeling Tool. 2014. Disponible en: http://www.visual-paradigm.com/product/vpuml/editions/enterprise.jsp

13. **Lira Turriza, Jose Luis**. Algoritmos y lenguajes de programación. Campeche: Instituto Tecnológico Superior De Calkiní. 2007.
14. **Hernández Orallo, Enrique**. El Lenguaje Unificado de Modelado (UML). 2012
15. **Principales características de java**. Disponible en: http://personales.upv.es/rmartin/cursoJava/Java/Introduccion/PrincipalesCaracteristicas.htm.
16. **Torres, Sánchez Sara Marlen** (docente pl 13 epigmenio gonzalez). Entornos de Desarrollo Integrado, 2013.
17. **NetBeans IDE**, 2016. Disponible: https://netbeans.org.
18. **Alegsa, Leandro**, "Definición de Framework". Alegsa.com.ar- Portal de informática, internet, tecnologías y web. 2010. Disponible en: http://www.alegsa.com.ar/Dic/framework.php
19. **Matos, Lenis Rodríguez y Sarmiento, Daniel Sastriques**, "Propuesta de un mecanismo de seguridad para el marco de trabajo Kairo". Tesis de Diploma, Universidad de las Ciencias Informáticas. 2012.
20. **Garzón, Ma. Teresa Pérez**, Bachillerato, F.P, "Sistemas Gestores de Bases de Datos". Revista Digital Innovación y Experiencias Educativas. Recuperado el, 2010, no 30, p. 2013. Disponible en: http://www.csi-csif.es/andalucia/modules/mod_ense/revista/pdf/Numero_30/TERESA_GARZON_1.pdf
21. **PostgreSQL**. The PostgreSQL Global Development Group, 2016. Disponible en: http://www.postgresql.org/about/
22. **Pressman, Roger S**. Ingeniería del Software. Un enfoque práctico. Sexta Edición. s.l.: Mc Graw Hill, 2003. 970-10-5473-3.
23. **Zapata, Javier**. Niveles de prueba del software. Niveles de prueba del software. 2013.
24. **Peña, Juan Manuel Fernández**. 2010. Pruebas de Software. 2010. Disponible en: www.uv.mx/personal/jfernandez/files/2010/07/Cap3 - Caminos.pdf.
25. **Sommerville, Ian**; Sawyer, Pete. Requirements engineering: a good practice guide. John Wiley & Sons, Inc., 1997.
26. **Letelier, Patricio**. Metodologías ágiles para el desarrollo de software: Extreme Programming (XP). 2006.

27. **Reynoso, Carlos Billy**. Introducción a la Arquitectura de Software. Universidad de Buenos Aires, 2004, vol. 33.
28. **Tabares, Ricardo Botero**. Patrones Grasp y Anti-Patrones: Un Enfoque Orientado a Objetos desde Lógica de Programación. Entre Ciencia e Ingeniería, 2011, no 8, p. 161-173.
29. **Riquenes, Josué Rivera**, et al. Subsistema Medida Cautelar para el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos. Serie Científica-Universidad de las Ciencias Informáticas, 2016, vol. 9, no 1.
30. **Pressman, Roger S**. "Ingeniería Del Software, Un Enfoque Práctico", Cuarta Edición. Ed. Mc. Graw Hill. 1997

GLOSARIO DE TÉRMINOS

Seguridad Informática: se define como SI al conjunto de métodos y herramientas destinados a proteger los bienes (o activos) informáticos de una institución y tiene el objetivo de garantizar su confidencialidad, integridad y disponibilidad [1].

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 [2].

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 existente en cada empresa para conseguir la eficacia exigida en el marco de la organización correspondiente [3].

Auditorías de Seguridad Informática: las auditorías de seguridad informática tienen como principal función, analizar los procesos relacionados únicamente con la seguridad, la cual puede ser física, lógica y locativa, pero siempre orientada a la protección de la información. Estas auditorías representan un conjunto de técnicas, actividades y procedimientos destinados a analizar, evaluar, verificar y recomendar en asunto relativos a la planificación, control, eficacia, seguridad y adecuación del ejercicio informático en una empresa, por lo que comprende un examen metódico, puntual y discontinuo del servicio informático, con vista a mejorar la rentabilidad, seguridad y eficacia. La misión que tiene la auditoria informática es la detención de fraudes o errores en la informática mediante los controles oportunos [4].

Políticas de Seguridad Informática: Las PSI tienen por objeto establecer las medidas de índole técnica y de organización, necesarias para garantizar la seguridad de las tecnologías de información (equipos de cómputo, sistemas de información, redes (Voz y Datos)) y personas que interactúan haciendo uso de los servicios asociados a ellos [5].

ANEXOS

Anexo 1: Acta de aceptación de los requisitos.

Acta de aceptación

ACTA DE ACEPTACIÓN

En cumplimiento del desarrollo de la Tesis: **Sistema informático para la realización de auditorías de Seguridad Informática en CEGEL**. Se hace entrega de los productos que se relacionan a continuación:

- Especificación de Requisitos de Software
- Historias de Usuarios

La Parte Cliente, luego de haber revisado los productos de trabajo determina que los mismos cumplen con los estándares establecidos para el diseño y redacción de estos artefactos, quedando aceptados de esta forma.

Entrega	**Recibe**
Nombre y apellidos: Verónica Hernández Urgelles	**Nombre y apellidos:** Ing. Yordanis García Leiva
Cargo: Estudiante	**Cargo:** Especialista en temas de SI en CEGEL

Fecha: 11/03/2016

Anexo 2: Acta de aceptación del sistema.

Acta de aceptación

ACTA DE ACEPTACIÓN

En cumplimiento del desarrollo de la Tesis: **Sistema informático para la realización de auditorías de Seguridad Informática en CEGEL.** Se hace entrega del producto:

- Sistema informático para la realización de auditorías de Seguridad Informática en CEGEL.

La Parte Cliente, luego de haber revisado el software, considera que la solución cumple con cada uno de los requisitos que originaron su desarrollo y constituye una herramienta de apoyo para el desarrollo de auditoría de seguridad informática en CEGEL.

Entrega	**Recibe**
Nombre y apellidos: Verónica Hernández Urgelles	**Nombre y apellidos:** Ing. Yordanis García Leiva
Cargo: Estudiante	**Cargo:** Especialista en temas de SI en CEGEL

Fecha: 22/06/2016

Anexo 3: Acta de liberación del sistema por el grupo de calidad de CEGEL.

UCI Universidad de las Ciencias Informáticas | FACULTAD # 3
CENTRO DE GOBIERNO ELECTRÓNICO

CEGEL

Acta de Liberación Interna de Productos Software

Fecha de emisión del acta: 23/06/2016

Emitida a favor de: Sistema informático para la realización de auditoría de Seguridad Informática en CEGEL.

Datos del producto

Artefacto	Versión	Estado final	Cantidad Iteraciones	Tipos de pruebas realizadas	Fecha de liberación
App: Sistema informático para la realización de auditoría de Seguridad Informática en CEGEL	1.0	0	2	Evaluación dinámica Pruebas de Funcionalidad	23/06/2016

Ing. Yordanis García Leiva
Asesor de Calidad CEGEL

Verónica Hernández Urgelies
Autor

Ing. Nailin Pérez Martínez
Responsable de la liberación

1