FACULTAD 3

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

GETET: herramienta para la gestión de expedientes técnicos de las estaciones de trabajo del CEIGE.

Autores: Marielys Garcia Echevarria.
Reynier Bauta Camejo.

Tutores: Ing. Ariam Jorge Pedrosa González.
MsC. Orlando A. Valenzuela Aguilera.

La Habana, 2015
Año del 57 Aniversario de la Revolución

Nunca permitas que te desalienten los que no creen en ti, al contrario, esfuérzate más y muéstrales que eres capaz. El éxito no depende de la suerte, sino de la determinación y el sacrificio.

Declaro que somos los únicos autores del trabajo titulado:

"GETET: Herramienta para la Gestión de Expedientes Técnicos de las Estaciones de Trabajo del CEIGE".

Autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmamos la presente en el mes de _____ del año _____.

_____ _____

Marielys García Echevarría. Reynier Bauta Camejo

_____ _____

MSc. Orlando A. Valenzuela Aguilera. Ing. Ariam Jorge Pedrosa González

A mis padres, hermano y el resto de mi familia; motores impulsores de mis logros y los motivos de cada uno de mis sacrificios.

Marielys.

A la memoria de mis abuelas Reyna y Amalia, que ya no están entre nosotros y no pudieron ver a otro de sus nietos hecho ingeniero, pero seguro se sentirían orgullosas de mí.
A mis padres, por toda una vida de enseñanzas y amor.
A mi hermano, por estar ahí cada vez que lo necesito.

Reynier.

Marielys:

Gracias a mamá, papá, hermano y sobre todo a diosito por darme la dicha de tenerlos día a día junto a mí, dándome las fuerzas necesarias para llegar a donde estoy hoy, obteniendo un título de ingeniera en ciencias informáticas, gracias por confiar en mí cuando todo parecía perdido y les dije no, yo seguiré, porque sé que si puedo y se los demostraré y a pesar de todos mis tropiezos siempre pude levantarme y seguir adelante porque son ustedes la razón de mi ser y a pesar de no estar juntica a ustedes durante todos estos años, siempre han estado en cada minuto de mi vida y lo estarán esté yo donde esté, los amos muchísimo.

Al resto de mi familia le agradezco por la preocupación y dedicación que siempre me han dado sobre todo mi abuela Yaya, gracias por estar siempre ahí, a mi abuela Yiya, mi dos abuelos, mis primas, mis tía Mariolys y Mirta, no hay palabras para describirlas y a mi gordi chiquitica Leyanita por ser tan inteligente.

A Evelio por compartir y hacer suyos los momentos más difíciles de mi carrera, por ser tan especial.

Agradecerle a mi tutor y amigo del alma, Ariam de todo corazón por todas las horas de sueño robadas, consejos, dedicación y empeño para que todo me saliera bien.

Yaicel, gracias por sopórtame todo este tiempo, comprenderme y siempre estar dispuesto a todo por mí a pesar de las circunstancias.

A mi amigo Ariel, gracias por aguantar todas mis malcriadeces, por estar siempre cuando te necesito y por tener los mismos malos hábitos en la comida.

A mis compañeros de aula en especial a Yunior, Roli, a Jose Ángel por ser el profe de todos, a Sosa a pesar de su mal genio, Negrín por siempre levantarme los ánimos, a Ale por ser el de mayor experiencia, a mi compañero de tesis Bauta a pesar de casi siempre estar peleados, los gima, gracias a todos por tanta paciencia conmigo.

A mis compañeras de cuarto Odalys, Arai y Alicia, fueron mis hermanas en todo este tiempo.

A mi amigo Luis Ángel por permitirme molestarte tanto desde primer año.

A todos los profes que contribuyeron a formarme como una profesional, especialmente a Sandro, gracias por darme tu apoyo cuando lo necesité.

A todos los profesionales del departamento Desarrollo de Componentes, Rene, Katia, Claudia, Orlando e Inoelkis, gracias por el tiempo dedicado y todos los buenos consejos.

Reynier:

Quiero agradecer a mis padres por entregarme su amor día a día, por toda su dedicación, por todo lo que he logrado hasta hoy. A mi hermano, por no cansarse nunca de guiarme, por estar presente siempre y ser mi inspiración.

A mis abuelas Reyna y Amalia, que siempre las llevo en mi corazón.

A mi abuelo Rodrigo por tener el corazón más joven de la familia y llenarnos siempre de su felicidad. A mi abuelo René por enseñarme que hay que ser fuerte en la vida.

A Luisi, el mejor de los primos y mi hermanito menor, gracias por tu confianza.

A ma Rebeca por su alegría. A tío Rodriguito por su amistad.

A mi tío Raúl y mi primo Raulito que hoy no están entre nosotros.

A Daylenis que también ha estado ahí cuando la he necesitado.

A mis primas Yane y Liset que a pesar de vernos poco y estar lejos siempre las llevo bien dentro, a mis primos Jesu y Yunior el dúo más loco de la familia, a la Masi, Yadira y Sandrita. A todos mis primos que son de primera.

A toda la familia.

A mis hermanos de la vida, Rogelio, Rosbel y Yordany gracias por su amistad. A todas aquellas personas especiales que han formado parte de mi vida aunque sea un instante. A todas las amistades de Buenaventura. A los vecinos del barrio. A mis hermanos de la UCT, los jimaguas, Yaicel, Rolando, Negrin. A mi compañera de tesis, a todos los que formaron parte del grupo en estos 5 años. A todas mis amistades de la universidad.

A todos los buenos profesores que he tenido a lo largo de mis estudios.

A la Universidad de las Ciencias Informáticas.

RESUMEN

En la actualidad el proceso de gestión y control de la información de las estaciones de trabajo del Centro de Informatización de Entidades se realiza de forma manual a partir del uso de la herramienta Microsoft Excel. Este engorroso proceso trae consigo la introducción de errores afectando además la calidad y rapidez con que se obtiene la información. Existen varias aplicaciones que se encargan de recopilar automáticamente toda la información del *hardware* de las estaciones de trabajo, pero algunas son privativas o no cubren todos los escenarios que el Centro de Informatización de Entidades necesita. Con el fin de satisfacer las necesidades del centro respecto al tema, se ha realizado un estudio de las aplicaciones que realizan reportes de inventarios de *hardware* para identificar posibles puntos de reutilización. Como resultado de esta investigación se obtuvo la implementación de una herramienta que permite la recolección y visualización de la información referente a las estaciones de trabajo y a los elementos más significativos relacionados con estas en una determinada área. Permite además la generación de los expedientes y fichas técnicas de cada una de las estaciones de trabajo.

Palabras clave: estación de trabajo, expediente técnico, ficha técnica, *hardware*.

ABSTRAC

The process of management and control of the workstations Center for Informatization Management Entities is currently done manually using Microsoft Excel. This cumbersome process and brings the introduction of errors. Also affect the quality and speed with which information is collected. Because of this there is the need for a tool to automate these actions. At present there are several applications that are responsible for collecting all this *hardware* information of the workstations but some are privative and do not cover all scenarios identified. In order to meet the needs of this department on the subject has conducted a study of applications that perform hardware inventory reports to identify possible points of reuse. As a result of this research, was obtained the implementation of a tool that will allow collection and display information of the workstations. It also allows the creation of files and technical sheet each workstation.

Keywords: workstation, technical file, technical sheet, hardware.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

La sociedad actual transita por una acelerada inmersión y difusión en todas sus esferas de las Tecnologías de la Información y las Comunicaciones (TIC). Tanto compañías, instituciones, hospitales y centros educacionales aplican las TIC con el fin de perfeccionar con mayor efectividad y rapidez las tareas y procesos que en ellos se realizan. Estas nuevas tecnologías son definidas según (Cobo Romaní, 2009) como *"herramientas que las personas usan para compartir, distribuir y reunir información y comunicarse entre sí o en grupos, por medio de las computadoras o las redes de computadoras interconectadas. Se trata de medios que utilizan tanto las telecomunicaciones como las tecnologías de la computación para transmitir información"*.

Cuba no está ajena a estas nuevas tendencias por lo cual apuesta por un abarcador programa de informatización de la sociedad, donde cada vez es mayor el número de instituciones que incorporan en sus procesos las nuevas tecnologías. Por tal motivo, algunas instituciones del país, especializadas en las ramas de la informática, trabajan cada día para incrementar el desarrollo de la industria del *software* dentro y fuera de la nación, aspirando a alcanzar una prestigiosa ubicación en el mercado internacional. Es importante además destacar que este desarrollo del *software* cubano está cimentado en el *software* libre, producto a las ventajas que este presenta sobre el *software* propietario. Lo anterior justifica la actual necesidad de que la población esté a los niveles de tales avances, tanto estudiantes como profesionales deben dirigirse a la búsqueda de nuevos métodos y soluciones para automatizar sus tareas y realizarlas de forma más sencilla, en el menor tiempo y con el menor costo posible sin disminuir la calidad final de las mismas (Carús Llera and Sánchez Santana, 2012).

La Universidad de las Ciencias Informáticas (UCI), fundada en el año 2002 con el propósito de convertirse en una ciudad digital avanzada, es una potencial entidad que hace énfasis en la producción y desarrollo de la industria del *software* cubano y la informatización del país, ayudando además a su economía (UCI, 2015). Esta joven universidad cuenta actualmente con varios centros de desarrollos especializados en las diferentes esferas de la sociedad, que abarcan desde la rama de la salud hasta la rama militar apoyando la defensa del país. Lo anterior justifica el hecho de que cada centro posea un número elevado de computadoras que sustentan el desarrollo productivo que en ellos se realiza.

En estos centros uno de los procesos vitales para mantener la seguridad y el control sobre las

estaciones de trabajo es el levantamiento de toda la información relacionada a las mismas para analizarla y verificarla en el momento que se desee, para lograrlo, se hace uso de un inventario. Los inventarios tienen como objetivo la fiscalización, identificación, y categorización de los recursos tangibles de las organizaciones. Una rama particular de los inventarios son los inventarios de red, estos son controles periódicos que se realizan al *hardware* y al *software* de los activos de una red de computadoras. Un sistema de gestión de inventario de *hardware* y *software* es una aplicación con soporte de datos, que acumula información sobre los activos informáticos en una red de ordenadores (Hernández Pérez, Ordoñes Leyva et al., 2013a; Vidal, 2012). Desde el punto de vista económico, los inventarios son de gran importancia en las empresas pues, al numerar los recursos, se obtiene una mejor visión de las necesidades y potencialidades de la organización sirviendo este conocimiento para apoyar la toma de decisiones.

El Centro de Informatización de Entidades (CEIGE), es uno de los centros existentes en la universidad que dirige su producción al desarrollo de aplicaciones web de gestión empresarial. Cuenta en la actualidad con 414 estaciones de trabajo distribuidas en cada una de sus áreas. Actualmente en este centro se lleva el control de la información de las estaciones de trabajo de forma manual. Haciendo uso de la herramienta *Microsoft Excel* se crea un registro que almacena el número de la estación, el número del medio básico y el número de serie, el local al que pertenece y la información relacionada con sus dispositivos (disco duro, placa base, memoria de acceso rápido, monitor, teclado, fuente, mouse, impresora, tarjeta de video, bocinas, escáner, tarjeta de red, chasis, microprocesador y unidad óptica). Además de cada uno de estos dispositivos se almacenan datos como la marca y el fabricante. Asimismo se lleva el control de los sellos de las estaciones registrando de los mismos, su número y además su sellador.

El levantamiento de esta información se hace consultando el S*etup* de cada estación de trabajo y actualizando el registro. Estos registros son enviados por correo electrónico al asesor de tecnología del centro y este consolida la información de todas las estaciones. Luego la envía a los jefes de áreas en su totalidad. Por tal motivo todos los jefes reciben la información de todas las áreas, haciendo tediosa la búsqueda de un área específica dentro de todo el cúmulo de información. Podría pensarse en una alternativa que sea capaz de filtrar toda esta información para garantizar que cada jefe reciba solamente la información relacionada a su área.

A partir de la información obtenida en el levantamiento se elaboran los expedientes y las fichas de las estaciones de trabajo. Se consulta el registro y para cada estación se llena una plantilla en la herramienta *Microsoft Word* con sus datos generales y técnicos. El volumen de información y lo engorroso de este proceso propician la introducción de datos erróneos. Estos errores generalmente se identifican cuando se imprime el expediente y la ficha de las estaciones, teniendo entonces que comprobar nuevamente la información de esa estación, actualizar el registro e imprimir el expediente y la ficha. El proceso de análisis de esta información se dificulta principalmente por la cantidad de datos que necesita el registro, influyendo negativamente en el tiempo de entrega de la información y la calidad de la misma. Además, la búsqueda y recuperación de datos es lenta afectando directamente, el proceso de toma de decisiones.

A partir de la situación problemática explicada anteriormente los autores de este trabajo identifican como **problema a resolver**: ¿Cómo facilitar la gestión de los expedientes técnicos de las estaciones de trabajo en el CEIGE para aumentar el control sobre el *hardware* de las mismas?

A partir del problema identificado se define como **objeto de estudio** las herramientas para la gestión de inventarios y **como campo de acción** las herramientas para la gestión de inventarios de *hardware*.

Para darle cumplimiento al problema identificado se establece como **objetivo general** de la investigación desarrollar una herramienta que permita la gestión de los expedientes técnicos de las estaciones de trabajo en el CEIGE para aumentar el control sobre el *hardware* de las mismas.

El objetivo general se desglosó en los siguientes **objetivos específicos**:

1. Elaborar el marco teórico de la investigación relacionado con el dominio del problema planteado para sentar las bases de la investigación.
2. Definir el análisis y diseño de la herramienta para la gestión de los expedientes técnicos de las estaciones de trabajo del CEIGE.
3. Implementar la herramienta para la gestión de los expedientes técnicos de las estaciones de trabajo del CEIGE.
4. Validar la solución mediante la aplicación de pruebas de caja blanca, caja negra y el método de pre-experimento.

Se propone como **idea a defender**:

Si se emplea una herramienta para la gestión de los expedientes técnicos de las estaciones de trabajo del CEIGE se aumentará el control sobre el *hardware* de las estaciones de trabajo.

Variable independiente:

Herramienta para la gestión de los expedientes técnicos de las estaciones de trabajo del CEIGE.

Variable dependiente:

Control sobre el *hardware* de las estaciones de trabajo.

Población: La población se define como el "conjunto de elementos que tengan una o más propiedades en común definidas por el investigador y que puede ser desde toda la realidad, hasta un grupo muy reducido de fenómenos" (León and González, 2008). Por tal motivo la población sobre la cual se estará trabajando se corresponde con las estaciones de trabajo de la universidad.

Unidad de estudio: son los "elementos, fenómenos sujetos o procesos que integran la población y pueden ser individuos, grupos de personas, hechos, procesos, talleres, turno de trabajo, empresas, documentos" (León and González, 2008). Por tanto se trabajará con la unidad de estudio las estaciones de trabajo del CEIGE.

Selección de la muestra y criterio de selección

Muestra: Las estaciones de trabajo del departamento Desarrollo de Componentes del CEIGE.

Criterio de selección: Para la selección de la muestra se utilizó la técnica no probabilística de muestreo intencional, con el objetivo de garantizar que al menos uno de los elementos seleccionados de la población tenga la característica deseada (Grau, Correa et al., 2004).

Durante el desarrollo de la investigación se utilizaron los siguientes **métodos científicos**:

Métodos teóricos:

- **Histórico y Lógico:** mediante este método se analiza la trayectoria real de los fenómenos, su evolución y desarrollo. Se utiliza este método ya que entre los objetivos propuestos está

identificar los procesos relacionados con la gestión de expediente técnico y reportes de *hardware* de las estaciones de trabajo del CEIGE.

- **Análisis y síntesis:** permite determinar si es factible o no utilizar las características o funcionalidades que aportan los sistemas estudiados para la gestión de expedientes técnicos de las estaciones de trabajo del CEIGE; resultado muy importante para enfocar la investigación en los elementos más significativos, arrojando ideas y conclusiones mucho más prácticas y concretas.
- **Modelación:** para representar de forma visual las características estructurales de la herramienta, así como el flujo de los distintos escenarios definidos en el diseño de la solución. Se modelan todos los procesos relacionados con la gestión de expediente técnico y reportes de *hardware* de las estaciones de trabajo del CEIGE.

Métodos empíricos:

Entrevista: es una conversación planificada entre el investigador y el entrevistado para obtener información. Se realizan entrevistas a los clientes para obtener información y entender mejor los procesos relacionados con la gestión de expediente técnico y reportes de *hardware* de las estaciones de trabajo del CEIGE.

Para una mejor comprensión, el presente documento se estructura en: resumen, introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos.

En el **Capítulo I** se realiza un estudio de los aspectos teóricos relacionados con la investigación. Contiene un análisis de las principales herramientas existentes encargadas de inventariar *hardware* y *software* en una red de computadoras. Conjuntamente se analizan y se proponen las herramientas y técnicas más apropiadas para el cumplimiento del objetivo general de la investigación.

En el **Capítulo II** se presenta la propuesta de solución explicando en detalle cada uno de sus elementos. También se enumeran los requisitos funcionales y no funcionales del *software* junto a los artefactos generados según la metodología de desarrollo seleccionada. Se describen las características del sistema así como su arquitectura, base de datos, patrones arquitectónicos y de diseño. En un último momento se realizan las conclusiones parciales del capítulo.

En el **Capítulo III** se analizan las pruebas y los resultados obtenidos con la propuesta de solución que se realiza en la investigación. Se describen además los principales artefactos generados durante las estrategias de pruebas utilizadas. En un último momento se presentan las conclusiones parciales del capítulo.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA:

1.1 INTRODUCCIÓN

En este capítulo se enuncian primeramente los principales conceptos teóricos que constituyen la base de la investigación efectuada para favorecer la comprensión del lector y el entendimiento del contenido de la investigación. En un segundo momento se realiza un estudio del estado del arte de las principales herramientas existentes que gestiona inventarios de *hardware* y *software*. Además se analizan las características fundamentales de las tecnologías, metodologías y herramienta de desarrollo que se utilizarán en el desarrollo de la propuesta de solución al problema inicial planteado. En un último momento se plasman las conclusiones parciales del capítulo.

1.2 CONCEPTOS FUNDAMENTALES

Con el objetivo de mejorar la comprensión del presente documento el presente epígrafe aborda los conceptos fundamentales relacionados con la investigación.

Según el diccionario Ilustrado Océano de la Lengua Española (RALE, 2001) se define **expediente** como el conjunto de todos los papeles correspondientes a un asunto o negocio. Procedimiento administrativo en que se enjuicia la actuación de alguien. Historial académico de un alumno, o laboral, de un funcionamiento o empleado. Partiendo de la definición anterior y asociándola a la rama de la informática se defina para esta investigación como **expediente técnico** el documento que registra la información asociada al *hardware* de una estación de trabajo, especificando además algunos datos generales relacionados al local donde se ubica la estación, No. serie, No. sello, responsable y estado.

Otro concepto relevante para la investigación es **inventario**, se define según el diccionario Ilustrado Océano de la Lengua Española (RALE, 2001) como la relación estimativa de los bienes y derechos que posee una empresa en un momento dado, y de las sumas que debe. Documento en que se expresan dichas relaciones. En la investigación se utilizará el término **inventario de *hardware*** que se refiere, basado en el concepto anterior, a la relación de todos los elementos de *hardware* de las estaciones de trabajo de un determinado local.

Otro término que se utiliza es **auditoría**, descrito según el diccionario Ilustrado Océano de la Lengua Española (RALE, 2001) como Inspección o verificación de la contabilidad de una empresa o una entidad, realizada por un auditor con el fin de comprobar si sus cuentas reflejan el patrimonio, la

situación financiera y los resultados obtenidos por dicha empresa o entidad en un determinado ejercicio. Refiriéndose en el contexto de la investigación, al control de una identidad a fin de comprobar que la información registrada en el inventario de *hardware* se ajuste con todos los datos de las estaciones de trabajo del local.

1.3 REVISIÓN DE ALGUNOS SISTEMAS DE OBTENCIÓN DE INVENTARIOS DE HARDWARE Y SOFTWARE

En este epígrafe se realiza un análisis de los sistemas de obtención de inventarios de *hardware* y *software* existentes y más utilizados en la actualidad según (DNA, 2012; Hernández Pérez, Ordoñes Leyva et al., 2013b; NetSupport, 2014; Ordoñes L, Avilés V et al., 2014). Estos sistemas son aquellos que tiene como propósito realizar la captura, procesamiento y almacenamiento de la Información de *hardware* y *software* en una red de computadoras. A continuación se realiza un análisis de estas aplicaciones con el objetivo de considerar si es favorable utilizar una de estas herramientas para solucionar el problema a resolver o desarrollar una nueva tomando como ejemplo las mejores funcionalidades de las existentes.

1.3.1 NETSUPPORT DNA

NetSupport DNA (*Dynamic Network Administration*) es la herramienta líder entre las privativas para la gestión de inventarios de red. Es capaz de detectar cambios en el *hardware*, *software*, energía y consumo de CPU de las computadoras inventariadas y soporta las plataformas GNU/Linux y Windows. Cuenta con funcionalidades para notificación de alarmas vía correo electrónico entre las que destacan las relacionadas con la información referente a la cantidad de espacio libre en discos. Permite la gestión y actualización de licencias de *software* e implementa la arquitectura cliente-servidor (DNA, 2012; Hernández Pérez, Ordoñes Leyva, Ramos Betancourt and León Rodríguez, 2013b).

Las principales deficiencias de estas herramientas es que no tiene soporte para los sistemas operativos Debian, Ubuntu y Nova, los más populares de la familia Linux en Cuba (NetSupport, 2014; Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014). No cuenta con funcionalidades para la configuración de otros tipos de incidencias pues los cambios a detectar no son modificables. Requiere del gestor de base de datos Microsoft SQL Server limitando su entorno de despliegue.

1.3.2 TOTAL NETWORK INVENTORY

En el caso de *Total Network Inventory* es un sistema de control de activos e inventarios de computadoras, diseñado para compañías medianas y grandes. *Total Network Inventory* interroga todos los ordenadores y portátiles en la red y genera la información sobre el sistema operativo, compilación, dispositivos, *software*, y procesos ejecutados en los equipos remotos. Soporta las plataformas GNU/Linux, MacOS y Windows. Permite la creación de breves informes que pueden ser impresos, visualizados en la pantalla y exportados a diferentes formatos (Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014).

Pese a sus potencialidades no es capaz de emitir alertas ante anomalías en los inventarios. El contenido de los inventarios es enviado en su totalidad por tanto no detectan los cambios ocurridos en los componentes (Hernández Pérez, Ordoñes Leyva, Ramos Betancourt and León Rodríguez, 2013b; Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014).

1.3.3 VEO

VEO se caracteriza por las acciones de control que puede tomar remotamente y las tareas distribuidas en las computadoras. Posee una arquitectura cliente-servidor usada para comunicar la consola de administración con las estaciones de trabajo. Puede realizar inventarios mediante tareas programadas. Permite tomar el control del ratón y el teclado, desbloquea la sesión del usuario, toma imágenes del escritorio de las computadoras, explora y transfiere archivos y establece tareas programadas. Sin embargo no permite crear un historial de inventarios, ni detectar cambios en los mismos. Solo es compatible con plataformas Windows, siendo una limitación en las empresas cubanas pues en varias se utilizan plataformas GNU/Linux. No posee ninguna estrategia de notificación de alertas (Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014; VEO, 2014).

1.3.4 OCS INVENTORY NG

En el mundo de las herramientas libres se destaca el uso de *OCS Inventory NG* que permite realizar reportes de los inventarios realizados. Soporta una amplia gama de plataformas, entre las que se destacan Windows y GNU/Linux. Permite la realización de inventarios de *hardware* y *software* e implementa la arquitectura cliente-servidor. Es capaz de enviar reportes a través del correo electrónico. Utiliza varias tecnologías como: PHP, Perl y MySQL. Esta herramienta no tiene implementaciones para la detección de cambios en los inventarios, ni para la emisión de alarmas

dada la ocurrencia de estos cambios (MODx, 2011a; MODx, 2011b; Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014).

1.3.5 CACIC

Otra de las aplicaciones de gestión de inventario de red en el mundo del *software* libre es CACIC (Configurador Automático y Colector de Informaciones Computacionales) este programa soporta las plataformas GNU/Linux y Windows. Es capaz de enviar alarmas a los administradores de sistemas cuando se identifiquen cambios en la ubicación física de los equipos, y en la configuración de los componentes de *hardware* de cada uno de los ordenadores. Utilizan varias tecnologías como: PHP, Python, DelphiTM y el gestor de base de datos MySQL. Su objetivo principal es minimizar el consumo de recursos en el proceso de obtención de los inventarios. Esta herramienta tiene como deficiencia que solo tiene soporte para el idioma portugués (Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014).

1.3.6 GRHS

La herramienta Gestión de Recursos de Hardware y Software (GRHS) fue desarrollada en la universidad. Esta herramienta permite definir incidencias sobre los cambios en el *hardware* y *software* de las computadoras. Estas incidencias pueden ser de tipo, componente, estado y nivel. El tipo se corresponde con el tipo de inventario que se realice, por ejemplo, *hardware* o *software*. El componente es la parte del inventario que puede tener cambios, por ejemplo, teclado para el tipo *hardware* o antivirus para el tipo *software*. El estado se refiere al tipo de cambio que puede ocurrir, por ejemplo, sustraído para el teclado o desactivado para el antivirus. El nivel representa una agrupación de incidencias según el nivel de seguridad que defina el usuario, por ejemplo, la desactivación del antivirus corresponde a un nivel crítico o la sustracción de un teclado a un nivel medio. Esta definición de incidencias permite que el usuario de forma sencilla establezca qué cambios, en qué componentes, constituyen una incidencia y el nivel de violación de seguridad que representa para la entidad (Ordoñes L, Avilés V, Hernández P and Rodríguez H, 2014).

Sin embargo, a pesar que GRHS pudiera ser una propuesta para dar solución al problema a resolver por la investigación, esta herramienta no realiza el inventario de dispositivos como la impresora y el escáner. Además no permite generar el expediente y la ficha técnica de las estaciones de trabajo.

Luego de analizar las principales herramientas de obtención de inventario de *hardware* y *software* se tomaron algunas funcionalidades que podrían reimplementarse en la solución deseada, por ejemplo a la hora de realizar la auditoría de *hardware* se concluyó que se podría utilizar la funcionalidad del OCS Inventory NG que permite la obtención del inventario de *hardware* de toda una subred. También se identificó que estos sistemas no permiten generar los expedientes técnicos y fichas de cada estación de trabajo, así como una serie de reportes y gráficos sobre el estado del inventario.

Incluso estos sistemas analizados carecen de una forma correcta de distribución de la información, enviándola a sus usuarios por correo electrónico sin importar el rol que poseen y sin la debida compartimentación de la misma. Además se puede añadir que en estos sistemas las estaciones de trabajo están asociadas a un usuario y no a una persona como tal, en el centro cada estación de trabajo está asociada, independientemente del rol y del usuario, a un responsable, quien interactúa directamente con esta. Por estas razones se decide desarrollar una herramienta nueva con las premisas de explotar al máximo las potencialidades de las tecnologías a utilizar, que sea simple y se pueda extender con facilidad.

1.4 METODOLOGÍA, TÉCNICAS Y HERRAMIENTAS UTILIZADAS EN LA SOLUCIÓN

1.4.1 METODOLOGÍA DE DESARROLLO

Con vistas a garantizar el correcto funcionamiento del desarrollo productivo, la Universidad de las Ciencias Informática adopta una nueva metodología por la cual se deben regir todos los proyectos productivos en la misma, se basa en una variación de la metodología “Proceso Unificado Ágil”(AUP, por sus siglas en inglés) en unión con el modelo CMMI-DEV v1.3. Esta metodología establece distintas fases y disciplinas por las que se debe transitar durante el desarrollo. Las fases definidas son: Inicio, Ejecución (Elaboración, Construcción, Transición) y Cierre. Las disciplinas son: Modelo (Modelado del negocio, Requisitos, Análisis y Diseño), Implementación, Pruebas (Internas, Liberación y Aceptación) y Despliegue (opcional) (Rodríguez Sánchez, 2014).

La solución que se desea implementar se enmarca en la fase de Ejecución, en la cual se ejecutan las actividades requeridas para desarrollar el *software*. Durante la Ejecución se realizan los procesos de negocio, se refinan los requisitos, se elabora la arquitectura y el diseño, se implementa y se libera el producto. El objetivo de esta fase es obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales (Rodríguez Sánchez, 2014).

Fases e iteraciones

Todas las disciplinas antes definidas (desde Modelado de negocio hasta Despliegue) se desarrollan en la Fase de Ejecución, de ahí que en las mismas se realicen iteración y se obtengan resultados incrementales. La siguiente figura es un esquema que representa dicho proceso.

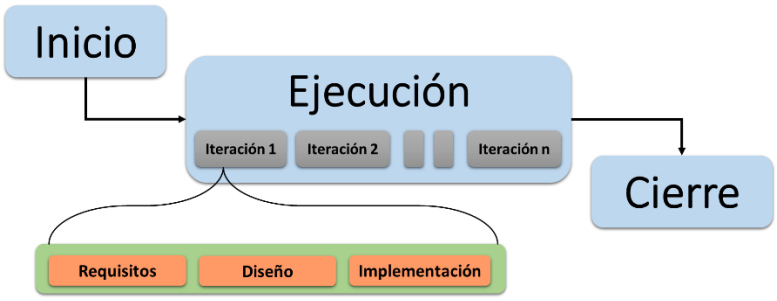


Figura 1: Fases e iteraciones de la Metodología de desarrollo AUP. Tomado de (Rodríguez Sánchez, 2014).

Durante la disciplina Modelado del Negocio se utiliza el modelo de proceso para comprender cómo funcionan los procesos de negocio que se desea informatizar y garantizar que el *software* desarrollado cumplirá su propósito. Además se emplea el modelado de domino, generando el modelo conceptual para favorecer a una mejor comprensión del problema, el cual describe los aspectos del dominio, identificándose los principales elementos físicos o lógicos del negocio y generalmente se presentan como clases. El modelo conceptual explica cuáles son y cómo se relacionan los conceptos relevantes en la descripción del dominio de un problema, identificando atributos y relaciones existentes entre ellos.

En la disciplina Requisitos se identifican los requisitos funcionales, luego se describen y posteriormente se elaboran los prototipos de interfaces de usuario asociados a ellos. Se identifican también los requisitos no funcionales que debe cumplir la herramienta También se genera el modelo de diseño, artefacto conformado por los diagramas de clases del diseño con estereotipos web y los diagramas de secuencia, en un último momento se realiza el modelo de datos y se elaboran los diseños de casos de prueba. En la disciplina Implementación se elabora el diagrama de componentes, se definen los estándares de codificación a utilizar y se implementan los requisitos funcionales identificados con sus correspondientes interfaces. En la disciplina Pruebas Internas se realizan pruebas de caja negra mediante la técnica de partición equivalente y las pruebas de caja blanca

utilizando la técnica del camino básico, también se resuelven las no conformidades detectadas durante la ejecución de estas pruebas.

1.4.2 INGENIERÍA DE REQUISITO

La Ingeniería de Requisitos se define como “el uso sistemático de procedimientos, técnicas, lenguajes y herramientas para obtener el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo que satisfaga las necesidades del usuario” (Pressman, 2005). Pressman agrega además que este proceso ayuda a los ingenieros de *software* a entender mejor el problema para el cual se necesita la solución y su objetivo es darle a todas las partes una explicación escrita del problema. Además incluye el conjunto de tareas que conducen a comprender cuál será el impacto del *software* sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el *software*.

Técnicas de captura de requisitos

Para hacer el proceso de captura de requisitos existen varias técnicas. La combinación de varias de ellas es una opción factible para lograr un buen proceso de captura de requisitos, y su uso está estrechamente relacionado con las características del proyecto en el que vayan a ser aplicadas. Las principales técnicas se relacionan a continuación.

Entrevistas: es un medio tradicional de obtención de requisitos. La entrevista es un método muy efectivo que permite conocer los problemas de los clientes y encontrar requisitos generales. Para aplicar este método es necesario conocer la forma en que se debe realizar una entrevista para lograr una buena comunicación entre el entrevistador y el entrevistado (Gunda Sai, 2008).

Prototipos: es la representación o visualización de parte del sistema. Es una herramienta valiosa para clarificar requisitos confusos. Proveen a los usuarios un contexto para entender mejor qué información necesitan proporcionar (Gunda Sai, 2008).

Tormenta de ideas: esta técnica es usada para generar nuevas ideas y encontrar la solución a cuestiones específicas. Es muy común en los comienzos del proceso de ingeniería de requisitos (Gunda Sai, 2008).

Técnicas de validación de requisitos

El proceso de validación de requisitos comprende actividades que generalmente se realizan una vez que se ha obtenido una primera versión de la documentación de requisitos. Según (Sommerville, 2005) la validación de requisitos "trata de mostrar que éstos realmente definen el sistema que el cliente desea. Su importancia radica en que los errores en el documento de requisitos pueden conducir a importantes costes al repetir el trabajo cuando son descubiertos durante el desarrollo o después que el sistema esté en uso".

A continuación se enunciaran varias de estas técnicas:

Generación de casos de pruebas: los requisitos deben poder probarse. Si las pruebas para éstos se conciben como parte del proceso de validación a menudo revela los problemas en requisitos. Si una prueba es difícil o imposible de diseñar, normalmente significa que los requisitos serán difíciles de implementar deberían ser considerados nuevamente (Sommerville, 2005).

Construcción de prototipos: consiste en mostrar un modelo ejecutable o semi-ejecutable a los usuarios finales y clientes para que experimenten con este modelo y valoren si satisface sus necesidades (Sommerville, 2005).

1.4.3 MODELO DE DISEÑO

El Modelo de diseño ha sido definido como "una abstracción del Modelo de Implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño" (Baryolo Gómez, 2010). El modelo de diseño puede contener: diagramas, clases, paquetes, interfaces, entre otros, que son utilizados como entrada esencial en las actividades relacionadas a la implementación.

Arquitectura de software

Según varios autores la arquitectura del *software* es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación. Es importante concebir una arquitectura sólida, pues la estructura del sistema depende en gran medida de la arquitectura de *software* que se utilice para desarrollarlo. Esta estructura se constituye de componentes, que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí (Camacho, Cardeso et al., 2004). La definición oficial ofrecida por la IEEE dicta que: "La Arquitectura del Software es la organización fundamental de un

sistema formado por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución" (IEEE, 2007).

En el diseño de la arquitectura de *software* se determina el estilo arquitectónico y el patrón arquitectónico, cuyas definiciones se muestran a continuación:

Estilo arquitectónico: expresa la arquitectura en el sentido más formal y teórico, describiendo una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro (Reynoso, 2004).

Patrón arquitectónico: expresa el esquema de organización estructural fundamental para sistemas de *software*. Este esquema provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Pudiera representarse como una plantilla para arquitecturas de *software* concretas, que especifica las propiedades estructurales de una aplicación (Buschmann, Kevlinn et al., 2007).

Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de *software*, o las relaciones entre ellos. Describe la estructura recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschmann, Meunier et al., 1996). A continuación se describen las dos vertientes principales de estos patrones:

Los Patrones GRASP: describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Tedeschi, 2012). Estos patrones incluyen algunos de los mencionados a continuación.

Experto: Asigna una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad (Larman, 1999).

Creador: Asigna a la clase B la responsabilidad de crear una instancia de clase (Larman, 1999).

Controlador: Asigna la responsabilidad de administrar un mensaje de eventos del sistema a una clase (Larman, 1999).

Bajo acoplamiento: Asigna responsabilidades de modo que se mantenga bajo acoplamiento entre clases (Larman, 1999).

Alta cohesión: Asigna responsabilidad de modo que se mantenga alta cohesión al objetivo de las clases que lo posean (Larman, 1999).

Asimismo existen los llamados **Patrones GoF**: se dieron a conocer a principios de los años 90 con el libro *"Design Patterns. Elements of Reusable Object-Oriented Software"*. Según su naturaleza se caracteriza en dicho libro a 23 patrones en los tres siguientes grupos (Prieto, 2009):

- Creacionales: inicialización y configuración de objetos.
- Estructurales: separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan para formar estructuras más grandes.
- Comportamiento: más que describir objetos o clases, describen la comunicación entre ellos.

1.4.4 MODELO DE IMPLEMENTACIÓN

El modelo de implementación se inicia a partir de los resultados obtenidos del diseño y constituye la modelación de los aspectos físicos del sistema a desarrollar. En él se describe cómo se implementan en términos de componentes los elementos del modelo de diseño que constituyen los planos lógicos del sistema. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado. Las dependencias entre los componentes, así como los recursos necesarios para poder ejecutar la herramienta desarrollada son aspectos que también se puntualizan en este modelo (Acuña, 2009).

Diagrama de componentes

En la modelación de los aspectos físicos de un sistema se realiza el diagrama de componentes, que muestra las dependencias entre un conjunto de componentes y se utiliza para modelar la vista de implementación estática. Gráficamente un diagrama de componentes es una colección de nodos y arcos, que normalmente puede contener componentes e interfaces, así como relaciones de dependencia, generalización, asociación y realización. Además pueden contener paquetes o subsistemas, los cuales se utilizan para agrupar elementos del modelo en bloques mayores (Jacobson, Booch et al., 1999).

1.4.5 PRUEBAS DE SOFTWARE

Las pruebas del *software* son un elemento crítico para la garantía de la calidad del *software* y representa una revisión final de las especificaciones, del diseño y de la codificación. Dichas pruebas son realizadas con el objetivo de detectar errores en el sistema, por lo que se llevan a cabo durante todo el ciclo de vida del producto. Los casos de prueba especifican una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, las condiciones bajo las que ha de probarse, así como los resultados esperados (Baryolo Gómez, 2010). Por otra parte Pressman asegura que además de detectar errores, las pruebas tienen como objetivo medir el grado en que el *software* cumple con los requisitos. Expone además que hay dos enfoques principales, las pruebas de "caja blanca" y las pruebas de "caja negra" (Pressman, 2005).

Pruebas estructurales o de caja blanca

También conocidas como pruebas de "caja de cristal", este es un método de diseño de casos de pruebas que utiliza la estructura de control del diseño procedimental para obtener los casos de pruebas que garanticen según (Pressman, 2005):

- Se ejerciten al menos una vez todas las rutas independientes del módulo.
- Se ejecuten todas las decisiones lógicas en sus opciones verdaderas y falsa.
- Se ejerciten todos los bucles en sus límites.
- Se usen las estructuras internas de datos para asegurar su validez.

Existen varias técnicas de pruebas de caja blanca como se describen a continuación:

Camino básico: permite 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. Para obtener el conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los casos de pruebas obtenidos garantizan que se ejecute al menos una vez cada sentencia del programa (Pressman, 2005).

Prueba de condición: ejercita las condiciones lógicas contenidas en el módulo de un programa y su propósito es detectar no solo errores en las condiciones, sino también otro tipo de errores (Pressman, 2005).

Pruebas funcionales o de caja negra

Este tipo de pruebas, también conocidas como “de comportamiento”, se concentran en los requisitos funcionales del *software* y descubren una clase diferente de errores a los que se descubren con los métodos de caja blanca (Pressman, 2005). Las pruebas de caja negra tratan de encontrar errores en las siguientes categorías:

- Funciones correctas o faltantes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y término.

Para realizar las pruebas de caja negra existen varias técnicas, algunas de ellas son las establecidas por (Pressman, 2005) y se mencionan a continuación:

Partición equivalente: es una técnica que divide el dominio de entrada de un programa en clases a partir de las cuales pueden derivarse casos de pruebas, permitiendo examinar los valores válidos e inválidos de estas entradas existentes en el *software*. Además descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. Esto permite reducir el número de casos de prueba a elaborar.

Análisis de valores límites: los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta es una técnica que complementa la partición equivalente y la mayoría de las veces se aplica de forma inconsciente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, lleva a la elección de casos de prueba en los extremos de la clase.

1.4.6. HERRAMIENTAS Y TECNOLOGÍAS PARA EL DESARROLLO DE LA PROPUESTA DE SOLUCIÓN

El éxito de los sistemas informáticos, depende en gran medida de una correcta elección de las herramientas y tecnologías a utilizar. Las definidas en la Vista del Entorno de Desarrollo Tecnológico de Sauxe se describen a continuación:

Lenguaje para el modelado

El Lenguaje Unificado de Modelado (UML) es un leguaje de modelado estandarizado de propósito general en el campo de la ingeniería de *software* orientada a objetos. UML incluye un conjunto de

técnicas de notación gráfica para crear modelos visuales de programación orientada a objetos (Pressman, 2005). Entre sus principales objetivos está lograr que el modelado visual sea independiente del lenguaje de implementación, consiguiendo así que los diseños realizados usando UML puedan implementarse en cualquier lenguaje que soporte las posibilidades de este; así como establecer un lenguaje estándar de comunicación entre todas las personas involucradas en el proyecto.

UML representa para los desarrolladores de aplicaciones y sistemas una serie de ventajas, al igual que para las organizaciones, entre estos beneficios destacan según (ENTEL, 2013):

- Produce un aumento en la calidad del desarrollo.
- Mejora en un 50% o más los tiempos totales de desarrollo.
- Brinda la posibilidad de obtener un "plano" del sistema.
- Ofrece un mejor soporte a la planificación y control del proyecto.
- Constituye un lenguaje de modelado entendido tanto por humanos como por máquinas.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y viceversa.

Lenguaje de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana (Mark, 2010). Son numerosos los lenguajes de programación existentes en la actualidad que se dirigen principalmente a dos tipos de *software*: aplicaciones web o aplicaciones de escritorio. Siendo estos utilizados convenientemente para desarrollo de sistema según sus propias características. La aplicación que se desarrollará en la presente investigación pertenece al grupo de las aplicaciones web por lo que comprende el uso de lenguajes de programación del lado del cliente y del lado del servidor.

Programación del lado del servidor

PHP 5.5 es un lenguaje interpretado de alto nivel, especialmente pensado para desarrollos web y el cual puede ser incrustado en páginas HTML. La mayoría de su sintaxis es similar a los lenguajes C, Java y Perl y es fácil de aprender. La meta de este lenguaje es permitir escribir a los programadores páginas web más dinámicas de una manera rápida y fácil (Gutmans, Bakken Stig et al., 2004). Este lenguaje de programación presenta muchas ventajas para los desarrolladores, dentro de las principales se pueden mencionar como expresa (Gutmans, Bakken Stig and Derick., 2004).

- Se caracteriza por ser un lenguaje muy rápido.
- Es independiente de plataforma, puesto que existe un módulo de PHP para casi cualquier servidor web.
- Compatibilidad con las bases de datos más comunes: PostgreSQL, MySQL, Oracle, Informix, entre otros.
- Presenta una gran potencia, un alto rendimiento, no consume muchos recursos y es seguro.

Programación del lado del cliente

Java Script 1.8 es un lenguaje de programación interpretado, utilizado principalmente en su forma del lado del cliente, permitiendo mejoras en la interfaz de usuario sobre todo en páginas web dinámicas. Es un lenguaje interpretado que lo soportan la mayoría de los navegadores como Internet Explorer, Netscape, Opera, Mozilla Firefox (ECMA, 2010).

Herramientas utilizadas

El servidor web **Apache 2.2** está desarrollado bajo el criterio OpenSource. Funciona en múltiples sistemas operativos, lo que lo hace prácticamente universal. Es un servidor altamente configurable de diseño modular. Entre sus principales características se encuentran (TheApache, 2013):

- Es multiplataforma.
- Presenta una alta configuración en la creación y gestión de logs. Apache permite la creación de ficheros de log a medida del administrador, de este modo se puede tener un mayor control sobre lo que sucede en el servidor.
- Se desarrolla de forma abierta.
- Extensible: gracias a que es modular se han desarrollado diversas extensiones entre las que destaca PHP.

El gestor de base de datos **PostgreSQL 9.1** es un sistema de gestión de bases de datos relacional con su código fuente disponible libremente. Utiliza un modelo cliente/servidor. Incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. Otras características que posee como menciona (Oracle, 2012) son:

- Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python, entre otros.
- Soporte de protocolo de comunicación encriptado por SSL.
- PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. similar al de Oracle, PL/SQL.

El **PgAdmin III** es una herramienta de código abierto para la administración de bases de datos PostgreSQL y derivados. Incluye una interfaz administrativa gráfica y una herramienta de consulta SQL. Es multiplataforma, libre y se puede adquirir el código fuente desde la web oficial. Permite desde ejecución de consultas SQL simples, hasta la elaboración de bases de datos complejas, con el apoyo de las últimas características de PostgreSQL. No requiere ningún controlador adicional para comunicarse con el servidor de base de datos. La aplicación se encuentra desarrollada por una comunidad de especialistas en base de datos de todo el mundo y se encuentra disponible en más de 30 idiomas (PgAdmin, 2011).

Subversión es una herramienta de código abierto multiplataforma para el control de versiones de ficheros electrónicos, como son el *software* o la documentación. Se basa en un repositorio central que actúa como un servidor de ficheros, con la capacidad de recordar todos los cambios que se hacen tanto en sus directorios como en sus ficheros. El repositorio incrementa un número global de revisión con cada conjunto de cambios enviados (commit). Es posible copiar y renombrar ficheros; crear una rama del proyecto es tan fácil como copiar un directorio. También se puede pedir una salida con las diferencias entre dos revisiones arbitrarias, o que recupere algún sub-árbol de la revisión N (Serradilla, 2008).

RapidSVN 1.6.6 es un cliente gráfico para Subversion. Es fácil de usar, tanto para los que conocen Subversion como para los que empiezan, pudiendo acceder a direcciones SVN, subir y descargar

contenido y sincronizarlo con el servidor original, comprobar su estado, crear y fusionar direcciones (López, 2010).

El **IDE NetBeans 7.4** es un entorno de desarrollo integrado de código abierto para desarrolladores de *software*. Cuenta con todas las herramientas necesarias para crear aplicaciones profesionales de escritorio, empresariales, web y aplicaciones móviles con la plataforma Java, así como con C/C++, PHP, JavaScript y Groovy. NetBeans recibe el soporte integrado para lenguajes dinámicos, todo en una herramienta de gran alcance. El editor de PHP de NetBeans ofrece plantillas de código y generación (getters y setters), la refactorización, información sobre herramientas de parámetros, consejos y soluciones rápidas, y la finalización de código inteligente. El IDE también ofrece un completo editor de HTML, JavaScript y CSS, con la ventaja de proveer un resaltado de sintaxis completo, completamiento de código inteligente y la comprobación de errores para HTML, CSS y JavaScript, incluyendo HTML 5, JavaScript 1.7 (Oracle, 2012).

Visual Paradigm (Visual-Paradigm, 2013) es una herramienta de Ingeniería de Software Asistida por Computación (CASE). Propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del *software* a través de la representación de todo tipo de diagramas. Entre sus principales características se pueden mencionar:

- Entorno de creación de diagramas para UML.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Disponibilidad de integrarse en los principales IDE.
- Generación de bases de datos.
- Importación y exportación de ficheros XML.

Frameworks y librerías para el desarrollo

Sauxe 2.3 es un marco de trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo (Baryolo Gómez, 2010). Siguiendo el paradigma de independencia tecnológica por el cual apuesta el país, reutiliza las siguientes tecnologías libres:

Ext JS 4.2 es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas que además de flexibilizar el manejo de componentes de la página como el DOM, Peticiones AJAX, DHTML, tiene la gran funcionalidad de crear interfaces de usuario bastante funcionales, fáciles de usar y atractivas, similar a una aplicación de escritorio. Esto permite a los desarrolladores web concentrarse en la funcionalidad de las aplicaciones web en lugar de las advertencias técnicas y las incompatibilidades respecto a los navegadores (Shea and Ramsay, 2008).

Su utilización como eje principal en el desarrollo, se deriva de las numerosas ventajas y asincrónicas al servidor. Sus características lo convierten en un potente framework, las cuales mejoran considerablemente la experiencia del usuario final de la aplicación, sin dejar de mencionar las herramientas que brinda para la creación de ventanas, formularios y validadores.

Zend Framework 1.11 se trata de un marco de trabajo para el desarrollo de aplicaciones web y servicios web con PHP, brinda soluciones para construir sitios web modernos, robustos y seguros. Además es libre y trabaja con PHP 5 (Zend-Technologies, 2013).

Doctrine 1.2.2 es un mapeador de objetos-relacional (ORM) escrito en PHP, es una capa de abstracción que se sitúa justo encima de un Sistema Gestor de Base Datos. Está completamente diseñado utilizando la técnica orientada a objetos, ofreciendo una capa de persistencia transparente a los objetos de PHP. Uno de sus puntos fuertes es que cuenta con su propio lenguaje de consultas a bases de datos llamado DQL, basado en el dialecto HQL de Hibernate (Doctrine, 2014).

Doctrine brinda la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa. Una de las ventajas que ofrece el uso de Doctrine, es que le evita al desarrollador escribir consultas SQL con sintaxis específicas para un sistema de gestión de bases de datos concreto, para ello transforma de forma automática las llamadas a los objetos en consultas optimizadas para el tipo de sistema que se esté utilizando (Doctrine, 2014).

1.5 CONCLUSIONES DEL CAPÍTULO

El desarrollo del presente capítulo permitió arribar a las siguientes conclusiones:

- Se realizó un estudio de las herramientas existentes de gestión de inventario de *hardware* permitiendo demostrar que estos sistemas no cumplen con las necesidades del cliente. Además se identificaron puntos de reutilización relacionados con funcionalidades y

características de estas herramientas que sirvieron como punto de partida para el desarrollo de la solución*.*

- Se llevó a cabo el estudio de la metodología, las herramientas y tecnologías para el desarrollo de la solución, evidenciándose que las mismas incluyen los requisitos necesarios para guiar el proceso de desarrollo. Además fue posible definir el entorno de desarrollo basado en las herramientas antes seleccionadas.

CAPÍTULO II: PROPUESTA DE SOLUCIÓN.

El presente capítulo se enfoca en la descripción de las características de la herramienta que propone la investigación para la gestión de expedientes técnicos de las estaciones de trabajo del CEIGE (GETET). Mediante la relación de aspectos teóricos y prácticos se establece un marco de trabajo a través del cual se determinan los principales componentes y artefactos del sistema. Algunos de estos elementos comprenden el modelado del negocio, los requisitos funcionales y no funcionales, la definición del modelo de datos y el establecimiento de patrones y estilos arquitectónicos, así como de diseño. Se relacionan además los principales artefactos generados por la metodología AUP.

2.1 DESCRIPCIÓN DE LA PROPUESTA DE SOLUCIÓN

Luego de analizadas las principales deficiencias que presenta actualmente el CEIGE en cuanto a la gestión de inventarios de sus estaciones de trabajo y las alternativas que existen en el mundo para solucionar algunas de ellas se propone desarrollar una herramienta que automatizará la gestión de los expedientes técnicos de las estaciones de trabajo de este centro. GETET permitirá generar los expedientes y las fichas técnicas de las estaciones de trabajo, garantizando una gestión centralizada de los recursos, aumentando la rapidez y efectividad del proceso. Además se podrán obtener algunos reportes sobre la información que gestiona el sistema tales como: la relación de las estaciones rotas o con sellos rotos, las estaciones disponibles y otros reportes relacionados a las prestaciones de las estaciones de trabajo. La mayoría de estos reportes se realizarán de manera gráfica con el propósito de facilitar el entendimiento por parte de los usuarios que interactuarán con la aplicación. La herramienta permitirá además obtener la información del *hardware* de las estaciones de una subred seleccionada y comparar esos resultados con los registrados en su base de datos, de esta manera se podrán detectar alteraciones en las características físicas de las computadoras y emitir alertas a usuarios ya predefinidos, informando de cualquier anomalía. GETET será capaz de personalizar la información conforme al área en la que se esté trabajando, de esta manera se resolverá el problema relacionado al cúmulo de información que recibían todos los jefes de áreas.

2.2 DESCRIPCIÓN DEL MODELO DE NEGOCIO

A continuación se presentan los diagramas de proceso de negocio del sistema utilizando BPMN (del inglés Business Process Model and Notation), para proveer una notación estándar que sea legible y entendible por parte de todos los involucrados e interesados en el negocio, en este caso los procesos

identificados y que posteriormente serán automatizados son: realizar inventario de *hardware* y realizar auditoría.

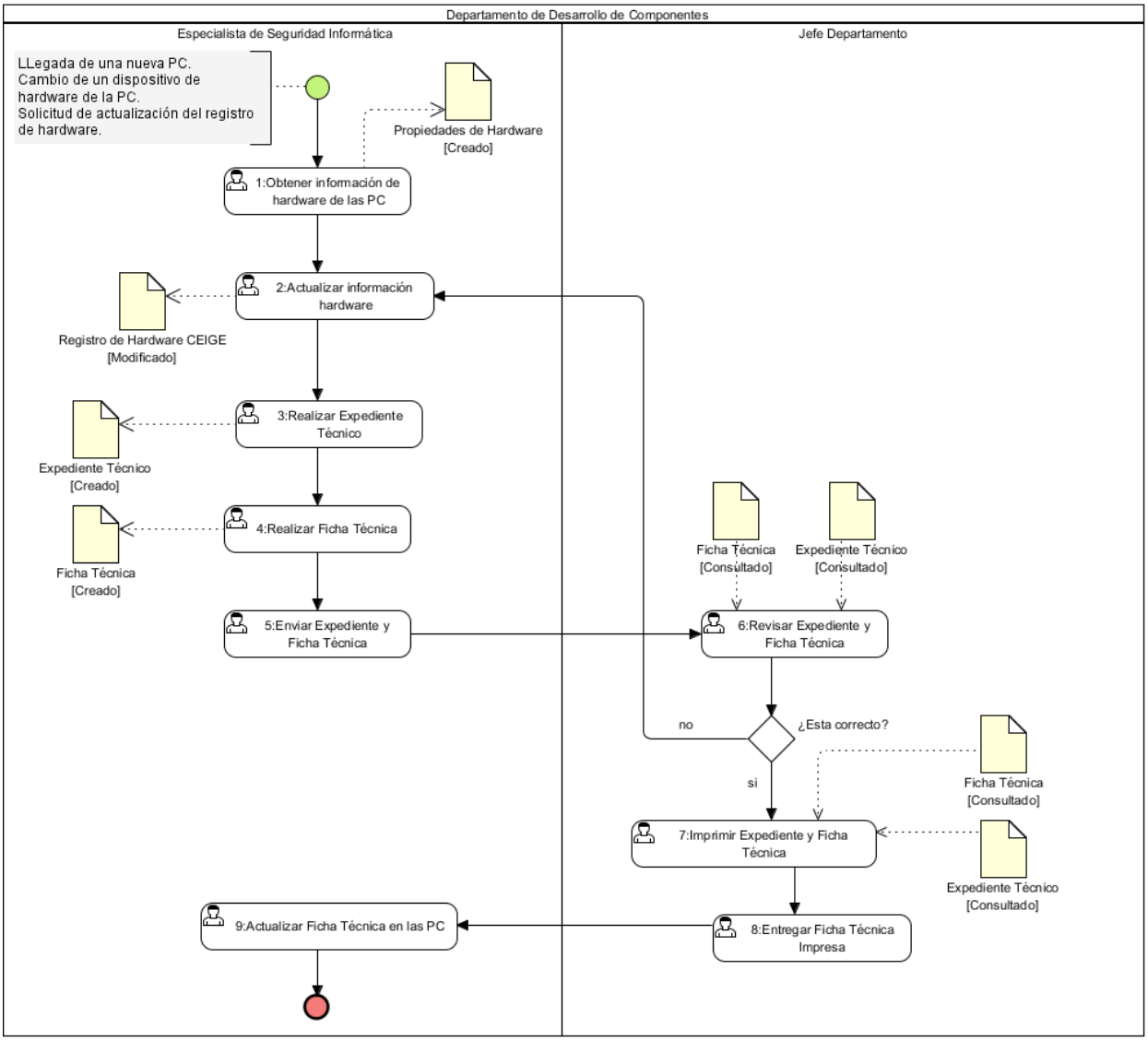


Figura 2: Diagrama de procesos de negocio: realizar inventario de *hardware*.

Como se muestra en el diagrama, el proceso de realización del inventario de *hardware* puede comenzar a partir de tres eventos fundamentales: llegada de una nueva computadora, cambio de un dispositivo de *hardware* de una computadora y la solicitud de actualización del registro de *hardware*. Luego se procede a la realización de varias actividades y el proceso culmina cuando se actualiza la

ficha técnica en las computadoras. En el artefacto que propone la metodología AUP, para la descripción de los procesos de negocio (***CIG-SXE-N-i1201.doc***), se encuentra una descripción más amplia de este diagrama.

El siguiente diagrama corresponde al proceso de negocio realizar auditoría.

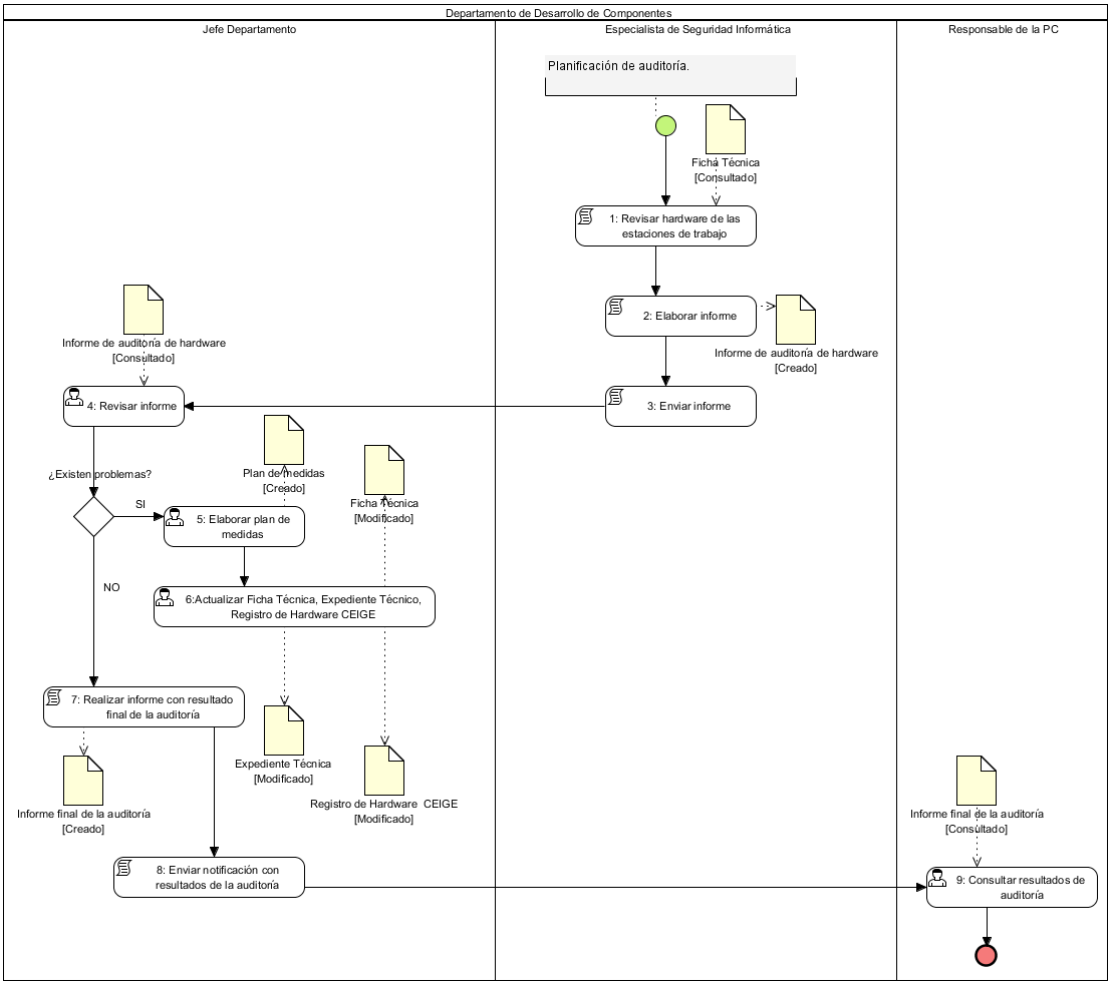


Figura 3: Diagrama de procesos de negocio: realizar auditoría.

Como se muestra en el diagrama, el proceso de realizar auditoría comienza a partir del evento planificación de auditoría. Luego se procede a la realización de varias actividades y el proceso culmina cuando se consultan los resultados de auditoría. En el artefacto que propone la metodología AUP, para la descripción de los procesos de negocio (***CIG-SXE-N-i1202.doc***), se encuentra una descripción más amplia de este diagrama.

2.3 MODELO CONCEPTUAL

El modelo conceptual se considera según Larman como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio. Mediante un lenguaje visual se pueden representar las cosas del mundo real que son de interés para el dominio de forma más sencilla y entendible (Larman, 1999). Lo anterior justifica la importancia y pertinencia de la realización de un modelo conceptual para la comprensión del funcionamiento de un entorno determinado y la relación entre sus conceptos fundamentales. A continuación se muestra el modelo conceptual de la propuesta de solución.

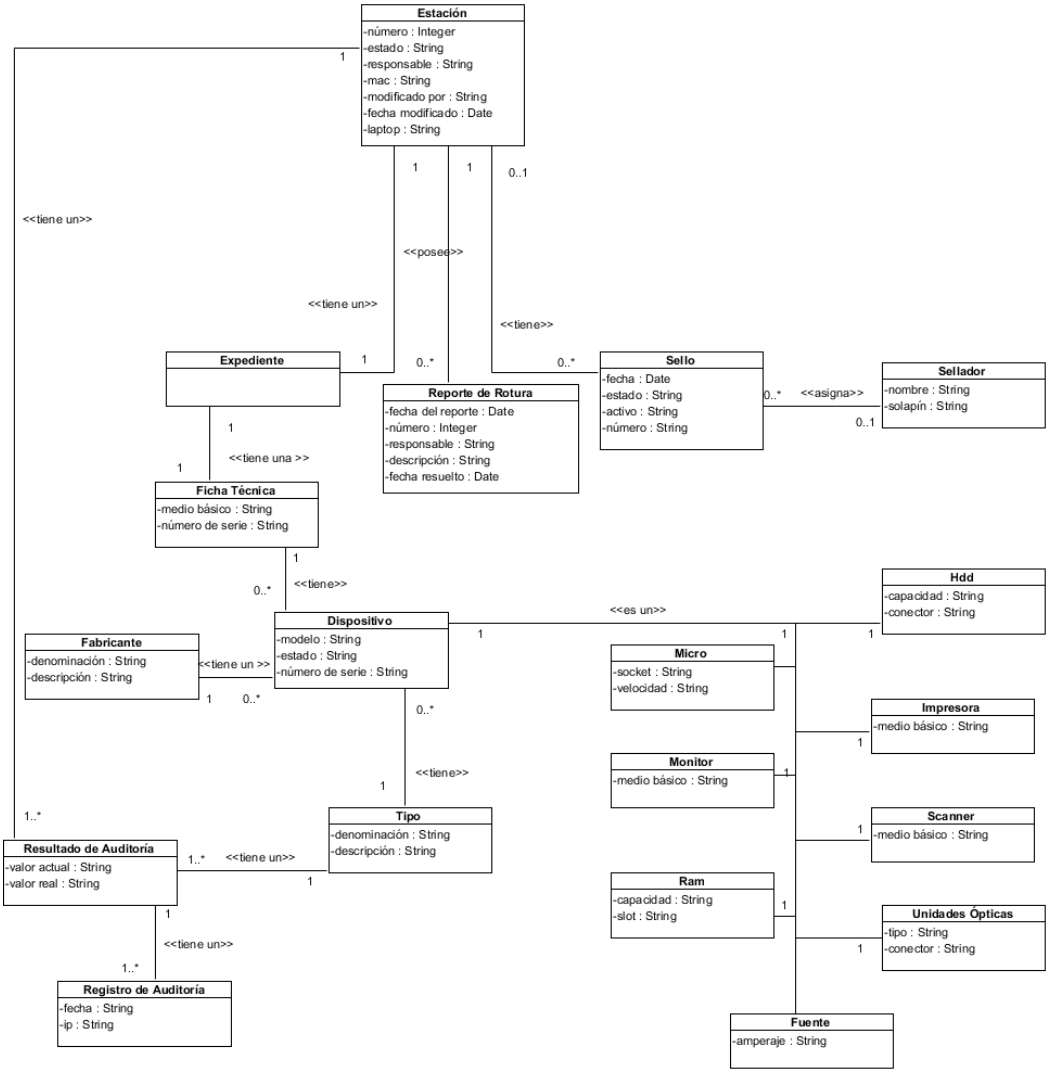


Figura 4: Modelo conceptual de la propuesta de solución

Como se observa en el diagrama existen 19 conceptos fundamentales, de estos conceptos los de mayor relevancia para la investigación lo constituyen **estación** y **expediente** los cuales se refieren a los datos generales de una estación de trabajo y a la información que se almacenará de dicha estación respectivamente. La explicación de los conceptos restantes se puede encontrar en el documento propuesto por la metodología AUP para la descripción del modelo conceptual (***CEIGE_GETET_Modelo_conceptual.doc***).

2.4 CAPTURA DE REQUISITOS FUNCIONALES Y NO FUNCIONALES.

La primera tarea que debe realizar un equipo de desarrollo de *software* en el instante de comenzar la captura de los requisitos de un sistema es determinar las técnicas que se aplicarán según el producto a desarrollar. Para implementar la propuesta de solución y en correspondencia con el entorno sobre el cual se desarrolla la investigación se utilizará la entrevista. Para aplicar esta técnica se realizaron reuniones donde estuvieron presentes cliente, los tutores y tesistas, determinando en conjunto los requisitos necesarios en dependencia de la necesidad del propio cliente.

Los requisitos funcionales se definen según (Sommerville, 2005) como las declaraciones de los servicios que debe ofrecer el sistema, su reacción ante entradas particulares y el comportamiento en situaciones específicas. En algunos casos los requisitos funcionales pueden declarar explícitamente lo que el sistema no debe hacer. Para GETET se determinaron los siguientes requisitos funcionales (RF):

AGRUPACIÓN DE REQUISITOS	RF	NOMBRE
MÓDULO GESTIONAR NOMENCLADORES		
Gestionar fabricante	RF 1.1	Adicionar fabricante
	RF 1.2	Modificar fabricante
	RF 1.3	Listar fabricante
	RF 1.4	Eliminar fabricante
	RF 1.5	Buscar fabricante
Gestionar sellador	RF 1.6	Adicionar sellador
	RF 1.7	Modificar sellador
	RF 1.8	Eliminar sellador
	RF 1.9	Listar sellador
Gestionar tipo de dispositivo	RF 1.10	Adicionar tipo dispositivo
	RF 1.11	Modificar tipo dispositivo
	RF 1.12	Listar tipo dispositivo

	RF 1.13	Eliminar tipo dispositivo en lote
	RF 1.14	Buscar tipo dispositivo
MÓDULO GESTIONAR EXPEDIENTE		
Gestionar estaciones de trabajo	RF 2.1	Adicionar estación de trabajo
	RF 2.2	Modificar estación de trabajo
	RF 2.3	Listar estación de trabajo
	RF 2.4	Cambiar ubicación de las estaciones de trabajo
	RF 2.5	Eliminar estación de trabajo en lote
	RF 2.6	Generar reporte de ficha técnica
	RF 2.7	Generar reporte de expediente técnico
	RF 2.8	Buscar estaciones de trabajo
Gestionar sellos	RF 2.9	Adicionar sello
	RF 2.10	Modificar sello
	RF 2.11	Listar sello
Auditoría	RF 2.12	Realizar auditoría
	RF 2.13	Mostrar resultados de auditoría
Gestionar dispositivos asociados a estaciones de trabajo	RF 2.14	Adicionar dispositivos
	RF 2.15	Modificar dispositivos
	RF 2.16	Eliminar dispositivos
	RF 2.17	Cargar dispositivos
	RF 2.18	Listar dispositivos
MÓDULO DE REPORTES		
Generar reportes	RF 3.1	Generar reporte de las estaciones de trabajo rotas
	RF 3.2	Generar reporte de las estaciones de trabajo con sello roto
	RF 3.3	Generar reporte de las estaciones de trabajo disponibles
	RF 3.4	Generar reporte de estaciones de trabajo con RAM DD3
	RF 3.5	Generar reporte de estaciones de trabajo con 1TB de disco duro
	RF 3.6	Generar reporte de estaciones de trabajo con 4GB de RAM
	RF 3.7	Generar reporte de estaciones de trabajo con micro Core i3
Generar gráfico	RF 3.8	Generar gráfico de máquinas rota
	RF 3.9	Generar gráfico de máquinas con prestaciones
Gestionar reportes de rotura	RF 3.10	Adicionar reporte de rotura
	RF 3.11	Resolver reporte

Tabla 1: Relación de los requisitos funcionales y sus agrupaciones.

Por su parte, los requisitos no funcionales (RNF) son aquellos que más allá de las funciones específicas del sistema, se refieren a las propiedades que emergen de este como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. En otras palabras, son aquellos requisitos que definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las posibles representaciones de los datos en las interfaces del sistema (Sommerville, 2005). Además

estos requisitos se clasifican según sus características técnicas en: requisitos de seguridad, de integridad, de disponibilidad, de usabilidad, de rendimiento, de portabilidad, de confiabilidad, requisitos de *software* y de *hardware*.

El desarrollo de la herramienta GETET forma parte del marco de trabajo Sauxe, desarrollado en el CEIGE. Por tanto, y de acuerdo con los requisitos que fueron establecidos por el centro al iniciar el proceso de desarrollo, los requisitos no funcionales a los que se debe acoger la aplicación a desarrollar son los que se describen a continuación:

Usabilidad: El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Rendimiento: Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 5 segundos para las actualizaciones y 20 para las recuperaciones.

Seguridad:

- Autenticación (Contraseña de acceso.)
- Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.
- La atención al sistema incluyendo, el mantenimiento de las bases de datos así como la salva de la información se realizarán de forma centralizada por el administrador.
- Verificación sobre las acciones irreversibles (eliminaciones).

Portabilidad: El sistema debe ser multiplataforma.

Soporte: La aplicación contará antes de su puesta en marcha con un período de pruebas para detectar posibles errores, se brindará el servicio de instalación, se garantizará la configuración inicial y se le dará mantenimiento una vez instalada.

Disponibilidad: El acceso a la información y a las funciones del sistema deberá estar disponible para los usuarios autorizados en cualquier momento que sea solicitada 24x7x365.

Requisitos de confiabilidad: Se garantizará la llegada de los datos de forma íntegra y segura al destino. Los datos serán almacenados de forma segura en una Base de Datos.

Requisitos de software: El servidor central de procesamiento deberá tener instalado como Sistema Operativo Windows o Linux, deberá contar además con un servidor de aplicaciones web como por ejemplo el servidor WAMP (Windows) o LAMP (Linux). El servidor central de bases de datos deberá tener instalado Sistema Operativo Windows o Linux y Gestor de Base de Datos PostgreSQL 9.2.

Requisitos de *hardware*: El Servidor central de procesamiento y el de base de datos requieren una memoria RAM de 2 GB DDR2 o superior, un procesador de 1.90 GHZ y 80 GB de capacidad de almacenamiento mínimo de disco duro. Además estos servidores deben poseer una tarjeta de red con velocidad mínima de transmisión de 1000 Mbps.

2.5 VALIDACIÓN DE REQUISITOS

Para lograr una correcta visualización de lo que se desea implementar en cada requisito y permitir que los clientes valoren si estos satisfacen sus necesidades, se utiliza la técnica de validación de requisitos: construcción de prototipos. Dentro de cada documento de descripción de requisitos se podrá observar el prototipo de interfaz correspondiente al requisito que se esté especificando. A continuación se muestra la especificación del requisito "*modificar estación de trabajo*". Los restantes documentos relacionados a la especificación de los requisitos del sistema se pueden encontrar dentro del expediente del proyecto.

Precondiciones	Se ha registrado al menos una estación de trabajo.
Flujo de eventos	
Flujo básico modificar estación de trabajo	
1	Se selecciona la estación de trabajo a modificar.
2	El sistema muestra y permite editar los datos de la estación de trabajo seleccionada.
3	Se introducen los datos de la estación de trabajo : Número Estado Responsable Medio básico Número de serie Laptop
4	El sistema valida (ver validación 1) los datos introducidos.
5	Si los datos son correctos el sistema los registra.
6	El sistema confirma el registro de los datos.
7	Concluye el requisito.
Pos-condiciones	

	Se modificaron los datos de la estación de trabajo.
Flujos alternativos	
Flujo alternativo 5.a Información errónea.	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 4 del flujo básico.
Pos-condiciones	
1	NA
Flujo alternativo 5.b Información incompleta.	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 4 del flujo básico.
Pos-condiciones	
1	NA
Flujo alternativo *.a El usuario cancela la acción.	
1	Concluye el requisito.
Pos-condiciones	
1	NA
Validaciones	
1	Ver referencia en el Modelo Conceptual
Conceptos	**Estaciones de trabajo** **Visibles en la interfaz:** Laptop Número Estado Responsable Medio básico Número de serie **Utilizados internamente:** N/A
Requisitos especiales	N/A
Asuntos pendientes	N/A

Tabla 2: Descripción textual del requisito: modificar estación de trabajo.

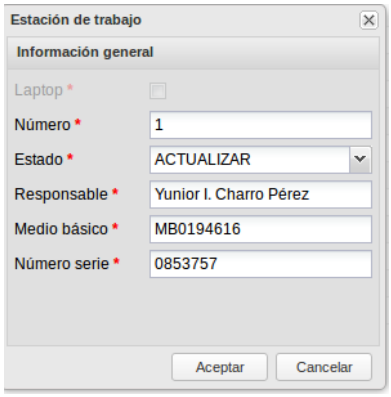


Figura 5: Prototipo de interfaz de usuario del requisito: modificar estación de trabajo.

2.6 MODELO DE DISEÑO

El modelo de diseño se define como “una abstracción del modelo de implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño” (Baryolo Gómez, 2010). El modelo de diseño puede contener: diagramas, clases, paquetes, interfaces, entre otros, que son utilizados como entrada esencial en las actividades relacionadas a la implementación. En el presente epígrafe se muestran los elementos que se tuvieron en cuenta para el diseño de la solución, así como los resultados y artefactos generados durante el diseño de GETET.

2.6.1 ARQUITECTURA DE SOFTWARE

Para el desarrollo del marco de trabajo Sauxe, en el departamento de Desarrollo Componentes del CEIGE se define el Estilo N capas, con cuatro capas fundamentales que se describen a continuación (Baryolo Gómez, 2010):

Capa de presentación: esta es la capa encargada de elaborar y mostrar las interfaces de los usuarios. La capa de presentación está compuesta principalmente por el marco de trabajo ExtJS y centra su desarrollo en tres componentes fundamentales: archivos JS, archivos CSS y páginas clientes y servidoras.

Capa de control o negocio: en esta capa se encuentra ubicado el marco de trabajo Zend, el cual implementa el patrón Modelo-Vista-Controlador.

Capa de acceso a datos: En la capa de acceso a datos está ubicado el marco de trabajo Doctrine, como ORM para la comunicación con el servidor de datos mediante el protocolo PDO.

Capa datos: en esta capa se encuentran ubicados los servidores de base de datos y los archivos XML donde se almacena la información de la aplicación.

En la capa de control o negocio de la arquitectura descrita anteriormente para Sauxe, se implementa el patrón arquitectónico MVC, el cual según (Baryolo Gómez, 2010) se encarga de separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes diferentes:

Modelo: compuesto por datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el encargado de persistir los datos: Doctrine.

Controlador: este componente gestiona las entradas al sistema y las respuestas que brindará al usuario.

Vista: gestiona las clases encargadas de mostrar la información del modelo al usuario.

2.6.2 PATRONES DE DISEÑO

Un patrón de diseño es definido por (Marquina, 2008) como un esquema de refinamiento de los subsistemas o componentes dentro de un sistema, o las relaciones entre estos. Describe una estructura común y recurrente de componentes interrelacionados, que resuelve un problema general de diseño dentro de un contexto particular. Los patrones de diseño tienen un alcance más reducido que los patrones de arquitectura y no dependen del lenguaje de programación, sin embargo la utilización de los mismos pudiera modificar la estructura de un subsistema. Los patrones de diseño están divididos en tres clases que son: Creacionales, de Estructura y de Comportamientos (Hamon, 2014) además existen también los conocidos Patrones Generales de Software para Asignar Responsabilidades GRASP (De sus siglas en inglés, *General Responsibility Assignment Software Patterns*) y los patrones GOF (De sus siglas en inglés, *Gang Of Four*) (Larman, 1999). Se describen a continuación los patrones utilizados en la implementación de GETET.

2.6.2.1 Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el *software* orientado a objetos (Visconti and Astudillo, 2006). De los

patrones GRASP, después de analizar las revisiones de (Larman, 1999; Visconti and Astudillo, 2006) se utilizan los siguientes:

El patrón **Experto** el cual se manifiesta en todas las clases del domino de la aplicación, por ejemplo, la siguiente figura muestra la clase **DatEstacion.php** experta en la información sobre las estaciones de trabajo, esta clase contiene la información necesaria para la creación de objetos o la implementación de métodos, como ***setUp()***, ***ObtenerTodos()***, para responder a una operación determinada.

```php
class DatEstacion extends BaseDatEstacion {

    public function setUp() {
        parent :: setUp();
        $this->hasMany('DatReporteRotura', array(
            'local' => 'id_estacion',
            'foreign' => 'id_estacion'
        ));
        $this->hasMany('DatExpediente', array(
            'local' => 'id_estacion',
            'foreign' => 'id_estacion'
        ));
    }

    static function ObtenerTodos($filter, $limit, $start, $local) {
        $filtro = "%" . strtolower($filter[0]->value) . "%";
        $result = Doctrine_Query :: create()->from('DatEstacion est')
                        ->where('est.id_local = ? and LOWER(est.responsable) LIKE ?', array($local, $filtro))
                        //->where( "LOWER(est.responsable) LIKE '$filtro'")
                        ->orderBy('est.id_local, est.numero')
                        ->limit($limit)
                        ->offset($start)
                        ->execute()->toArray();
        //print_r($local);die('ffff');
        return $result;
    }
```

Figura 6: Fragmento de código de la clase DatEstacion.php.

El patrón **Creador** para guiar la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Al escoger una clase como creadora, se da soporte al bajo acoplamiento. Se manifiesta en todas las clases del negocio de la aplicación, por ejemplo en la clase **DatEstacionModel.php** que se muestra en la siguiente figura, evidenciándose en la creación de objetos de las clases **DatExpedienteModel.phh** y **DatFichaTecnica.php**, para su posterior manipulación.

```php
        $estacion->save();

        $exp = new DatExpedienteModel();
        $id_expediente = $exp->AdicionarExpediente($estacion->id_estacion);

        $ficha = new DatFichaTecnica();
        $ficha->id_expediente = $id_expediente;
        $ficha->numero_serie = $mb;
```

Figura 7: Fragmento de código de la clase DatEstacionModel.php

Se utiliza también el patrón **Controlador** referido a asignar a una determinada clase la responsabilidad del manejo de los mensajes de los eventos de un sistema. Este patrón se manifiesta en todas las clases controladoras de la aplicación, por ejemplo en la clase **GstestacionController.php** encargada de manejar el flujo de datos entre la capa de presentación y la lógica de negocio del módulo Gestionar Expediente.

Figura 8: Fragmento de código de la clase GstestacionController.php

Bajo acoplamiento: El bajo acoplamiento se refiere a la escasa dependencia que debe existir entre las clases para lograr una mayor reutilización de ellas así como reducir el impacto de los cambios en el sistema y promover una mayor productividad. Si se analiza GETET, teniendo en cuenta además que en él se aplica el patrón Experto cada clase posee su lógica y solo ella la conoce, permitiendo así que otras clases solo tengan que solicitar alguna funcionalidad o servicio y mantener de este modo el bajo acoplamiento.

Este patrón se evidencia fundamentalmente en las clases de acceso a datos, pues estas no guardan relación alguna con las vistas y los controladores, el único modo de acceder a las mismas es por medio de las clases del dominio. Ejemplo de lo anterior se puede observar en el diagrama de clase **Gestionar dispositivos asociados a las estaciones de trabajo**.

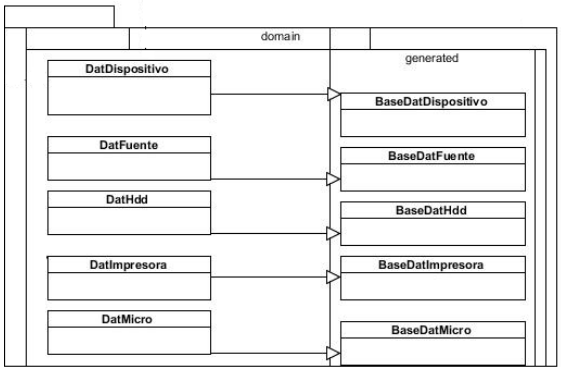


Figura 9: Fragmento del diagrama de clases Gestionar dispositivos asociados a las estaciones de trabajo.

Alta Cohesión: El patrón Alta Cohesión está referido a las clases que poseen responsabilidades específicas a un área funcional y colaboran con otras para llevar a cabo determinadas tareas. El uso de este patrón se evidencia en el fragmento del diagrama de clase **Gestionar tipo de dispositivos** que se muestra en la siguiente figura, evidenciándose que cada clase del sistema debe realizar solo las operaciones necesarias para cumplir una tarea dentro de un área funcional, manteniendo un número equilibrado de responsabilidades.

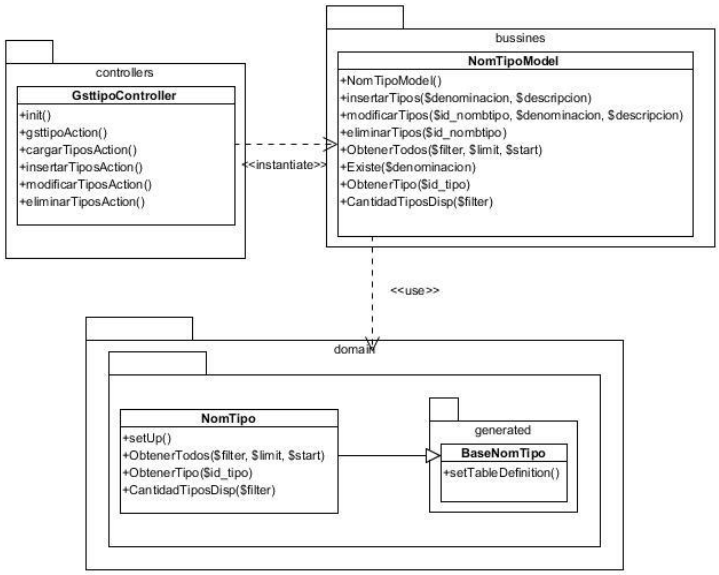


Figura 10: Fragmento del diagrama de clases Gestionar tipo de dispositivos.

2.6.2.2 Patrones GOF

El patrón **Apoderado/Proxy** se manifiesta en la intercomunicación existente de los módulos **Gestionar expediente** y el módulo **Nomencladores**, esta interacción se establece a través de servicios que responden a funcionalidades como: **GetNombreSellador, GetSelladores GetDenominacionFabricante, GetFabricantes, GetFabricanteReg, IdFabricante, GetTipo, GetTipos**. Haciendo uso de este patrón se evita importar las clases modelos de otros módulos para acceder a su información.

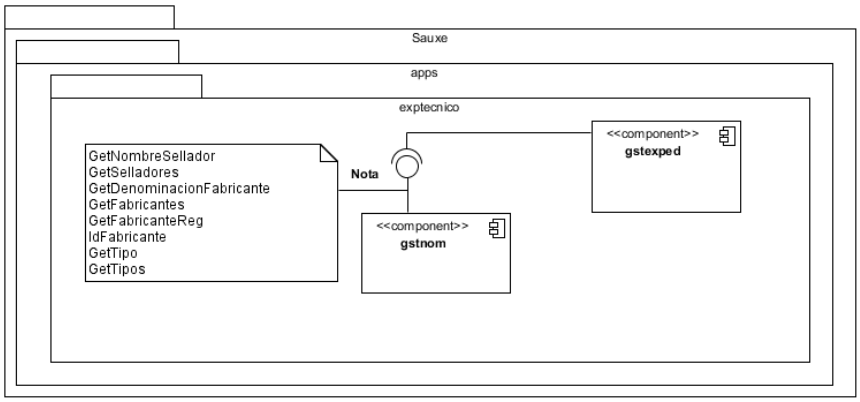


Figura 11: Fragmento del diagrama de componentes de GETET

Otro patrón utilizado en la aplicación es el **Acción/Command**. El uso de este patrón se manifiesta en la clase **EstacionTrabajoGrid.js**, mostrándose en las barras de botones y menús desplegables resaltados en rojo en la siguiente figura.

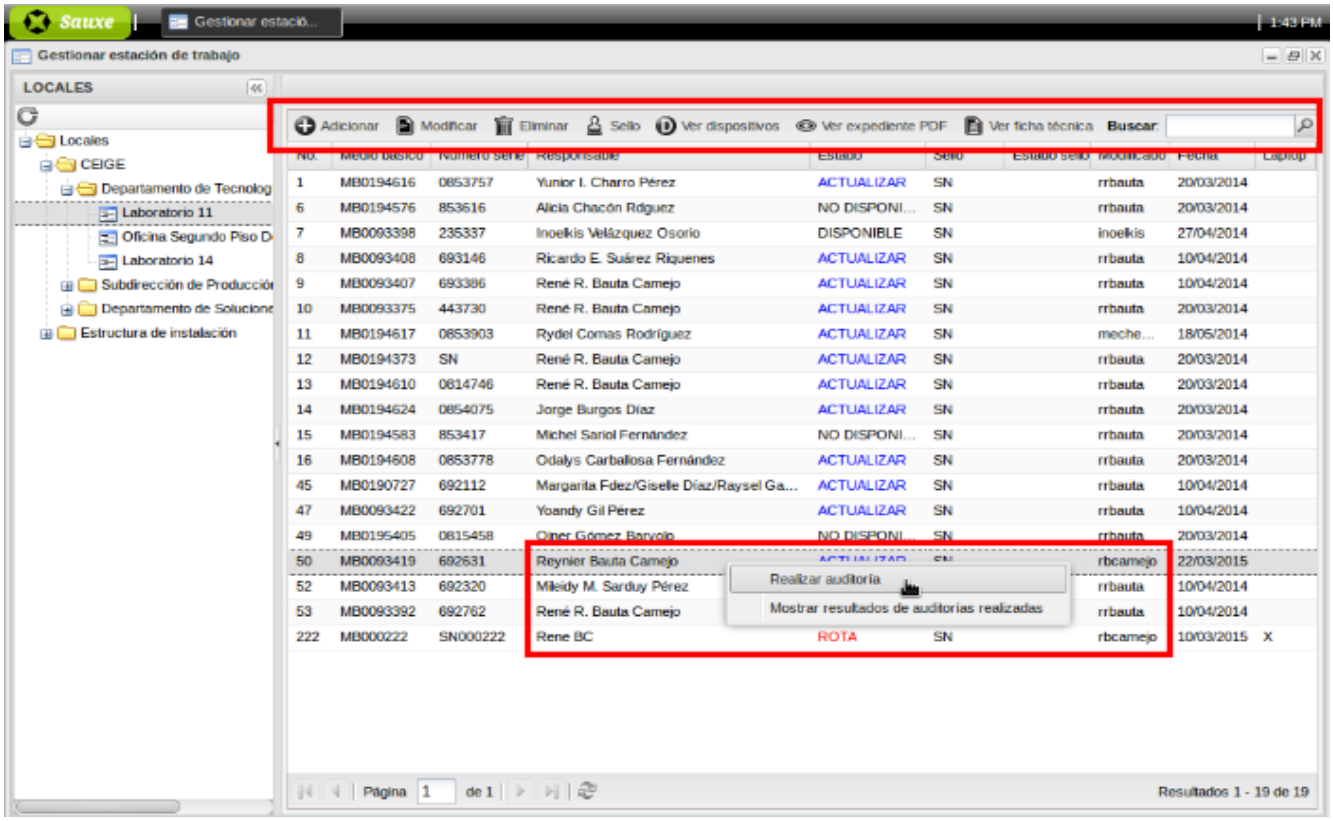


Figura 12: Interfaz de la clase EstacionTrabajoGrid.js

2.6.3 DIAGRAMA DE CLASES DEL DISEÑO CON ESTEREOTIPOS WEB

Los diagramas de clases especifican las diferentes clases que serán utilizadas en el sistema, así como las relaciones que existen entre ellas. Para GETET se elaboraron 8 diagramas de clase con estereotipos web. A continuación se muestra el diagrama de clase **Gestionar dispositivos asociados a las estaciones de trabajo**, para su mejor entendimiento solo se muestran en el diagrama los métodos de la clase controladora. En el Anexo 1 se puede encontrar el diagrama en su totalidad. Los restantes diagramas de clases pueden ser consultados en el expediente del proyecto.

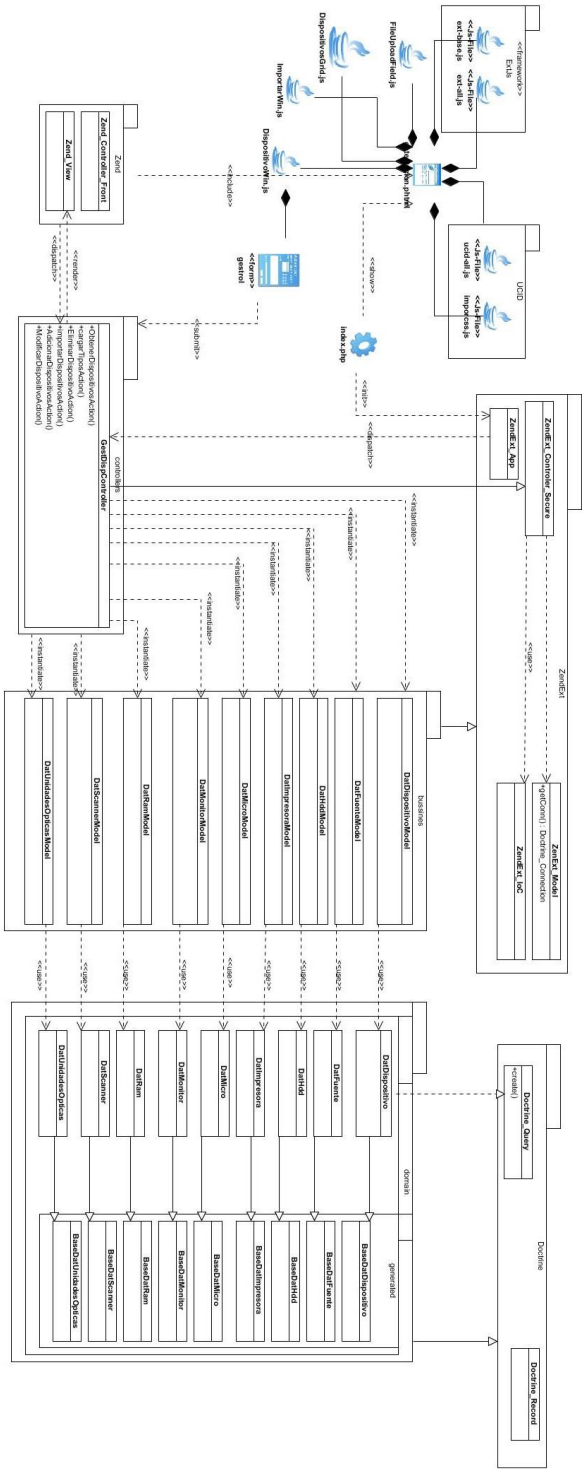


Figura 13: Diagrama de clases con estereotipos web para el proceso de gestionar dispositivos asociados a las estaciones de trabajo.

2.6.4 DIAGRAMA DE SECUENCIA

Los diagramas de secuencia describen la interacción entre los objetos de una aplicación y los mensajes recibidos y enviados por los objetos (ALTOVA, 2014). A continuación se muestra el diagrama de secuencia correspondiente al RF2.14: Adicionar dispositivos asociados a las estaciones de trabajo.

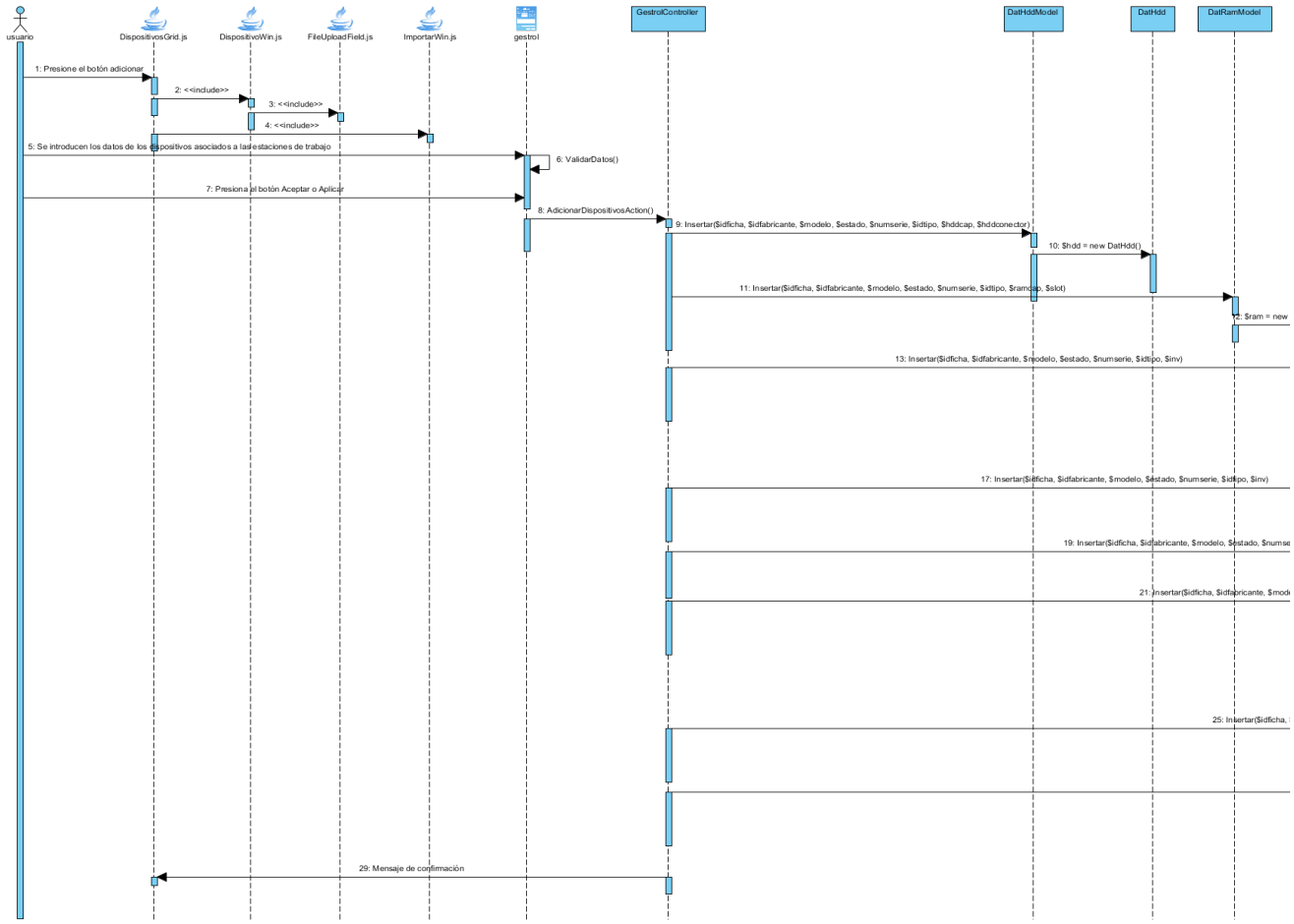


Figura 14: Fragmento 1 del diagrama de secuencia correspondiente al RF2.14: Adicionar dispositivos asociados a las estaciones de trabajo.

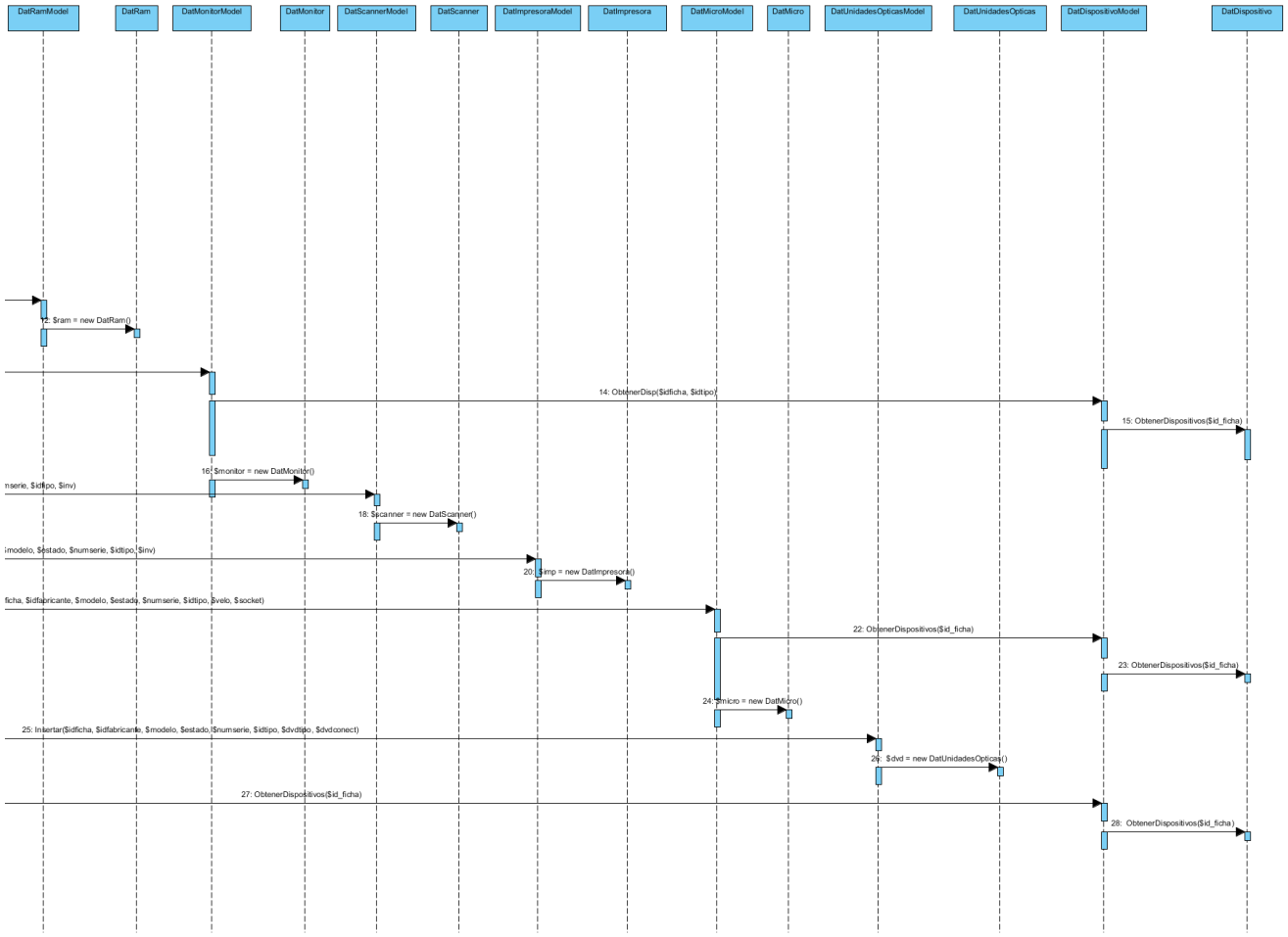


Figura 15: Fragmento 2 del diagrama de secuencia correspondiente al RF2.14: Adicionar dispositivos asociados a las estaciones de trabajo.

2.6.5 MODELO DE DATOS DEL SISTEMA

El modelo de datos es una definición lógica, independiente y abstracta de los objetos, operadores y demás que en conjunto constituyen la máquina abstracta con la que interactúan los usuarios. Los objetos permiten modelar la estructura de los datos y los operadores permiten modelar su comportamiento (Date, 2001). El modelo de datos correspondiente a GETET contiene 19 tablas que se muestran en la Figura 16.

.

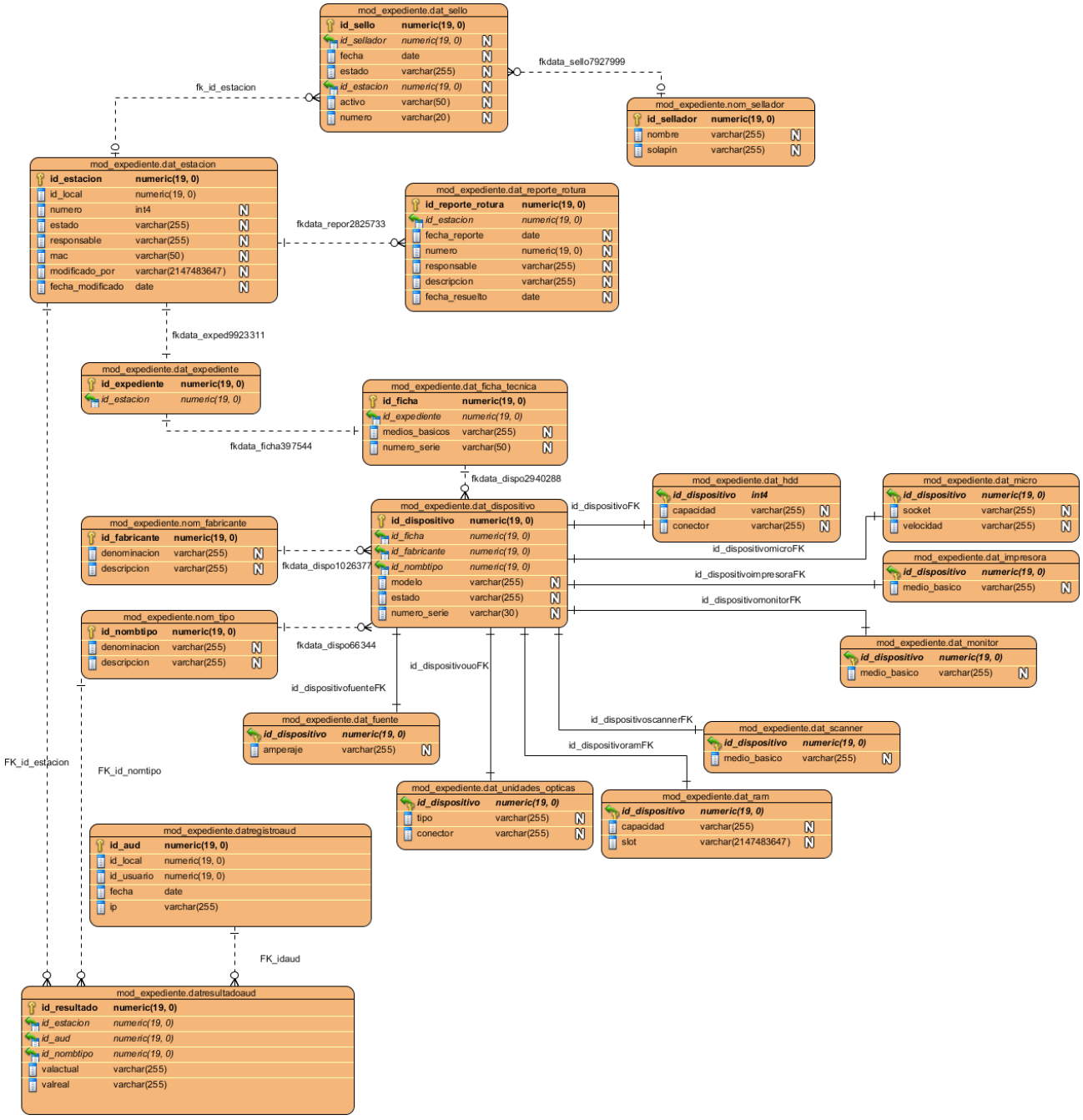


Figura 16: Modelo de datos de GETET.

Una de las tablas que más relevancia posee en el modelo de datos es **mod_expediente.dat_estacion** que contiene los siguientes atributos:

Id_estacion: llave primaria de la tabla.

Idlocal: referido al local donde se ubica la estación de trabajo.

numero: número que recibe cada estación de trabajo.

estado: estado en el que se encuentra la estación de trabajo (bueno, malo, regular).

responsable: almacena el nombre del responsable de la estación de trabajo.

mac: dirección física de la pc.

modificado_por: muestra el nombre del usuario que le realizó una modificación a la pc.

fecha_modificado: muestra la fecha en que se modificó algún cambio a la pc.

2.7 CONCLUSIONES DEL CAPÍTULO

Al concluir el presente capítulo los autores de la investigación arriban a las siguientes conclusiones:

- El modelo conceptual mostrado en el capítulo puntualiza los conceptos relacionados con el dominio del problema y su representación conforma una guía visual para la implementación de la solución tributando directamente a la fase de requisitos.
- La captura, especificación y validación de los 43 requisitos funcionales mostrados en 10 agrupaciones de requisitos generales, aportó una explicación escrita del problema y una mayor comprensión del comportamiento que se necesita implementar en la solución. Los requisitos no funcionales que debe cumplir la herramienta, garantizan que su desempeño es el adecuado para su integración en Sauxe.
- La realización del diagrama de clases del diseño con estereotipos web, los diagramas de secuencia y el modelo de datos, permitió representar: las clases y relaciones que componen el sistema; el flujo del comportamiento e interacción entre los objetos de la aplicación; y la estructura que tendrán los datos de la herramienta respectivamente, de forma tal que a través de estos diagramas se facilitó el entendimiento previo al proceso de implementación.

CAPÍTULO III: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

El siguiente capítulo aborda los elementos relacionados con la implementación de la Herramienta para la gestión de expedientes técnicos de las estaciones de trabajo del CEIGE, así como lo referente a las pruebas realizadas al sistema para validar su correcto funcionamiento. Se describen los estándares de codificación utilizados en función de garantizar el entendimiento del código fuente. Además se muestran los resultados de la aplicación de las métricas Tamaño Operacional de Clases y Relaciones entre Clases al diseño elaborado en el capítulo anterior.

Se utiliza la técnica del camino básico para obtener la complejidad lógica de los procedimientos. Se describe el proceso de confección de los casos de pruebas diseñados para realizar las pruebas funcionales, así como los resultados obtenidos, validando así que los requisitos identificados fueron implementados correctamente.

3.1 MODELO DE IMPLEMENTACIÓN

El modelo de implementación se inicia a partir de los resultados obtenidos en el diseño y describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado. También describe cómo dependen los componentes unos de otros además de los recursos necesarios para poder ejecutar la herramienta desarrollada (Acuña Brito, 2013).

3.1.1 DIAGRAMA DE COMPONENTES

Uno de los artefactos generados en esta fase es el diagrama de componentes el cual representa la estructura física del sistema, su agrupación por paquetes y las dependencias entre ellos. A continuación se muestra el diagrama de componentes de la herramienta GETET.

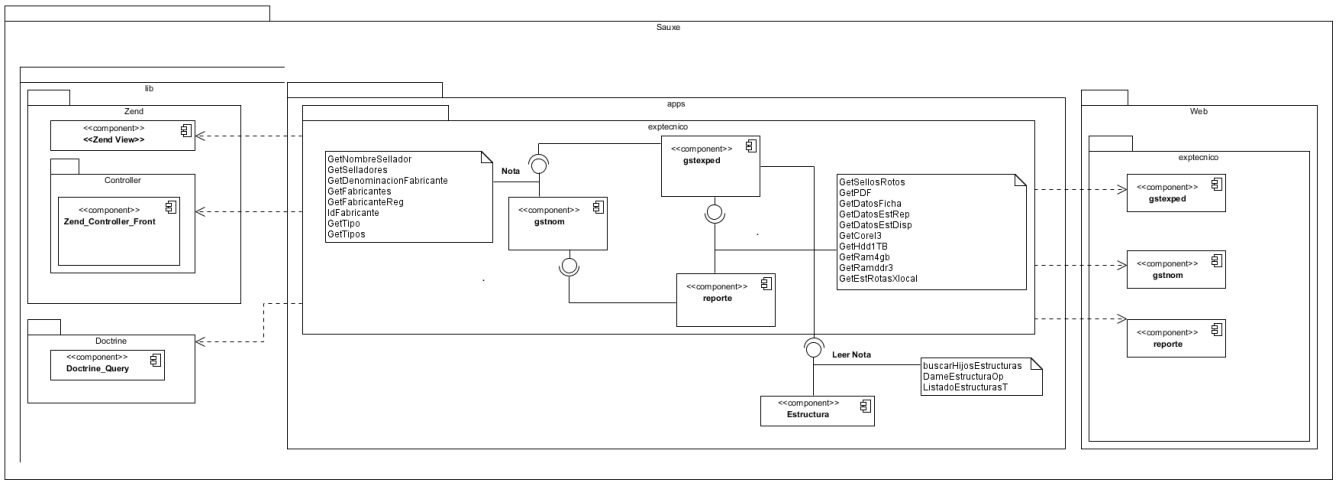


Figura 17: Diagrama de componentes de GETET.

3.2 ESTÁNDARES DE CODIFICACIÓN

Un factor importante para la implementación lo constituye la forma en que se construye el código fuente, esencialmente su organización y el estilo de codificación utilizado en el desarrollo. Una buena estructuración del código, mejora la lectura del *software*, permitiendo entender código nuevo rápidamente y más a fondo (Lobo, 2012). El uso de estándares de codificación permite lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo (Pérez Alfonso, 2012).

Para el desarrollo de la solución se utilizarán algunos de los estándares y normas de codificación propuestos como parte de la línea de arquitectura determinada para el proyecto Planeación de Recurso Empresarial (ERP) Cuba, los cuales se muestran a continuación:

Notación Húngara: definir prefijos para cada tipo de datos y según el ámbito de las variables. La idea de esta notación es la de dar mayor información al nombre de la variable, método o función definiendo en ella un prefijo que indique su tipo de dato o ámbito (Sperberg, 2012).

Notación PascalCasing: es similar a la notación húngara pero sin prefijos. En este caso los identificadores y nombres de las variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula (Svensk, 2012) .

Notación CamelCasing: es semejante al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula (Svensk, 2012).

A continuación se explican los estándares de codificación usados en GETET.

Nomenclatura de las clases

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing. Con sólo leerlo se reconoce su propósito. Ejemplo: DatDispositivo.

Nomenclatura según el tipo de clases:
Clase controladora: después del nombre de la clase se agrega la palabra “Controller”. Ejemplo: GstestacionController.

Clases del modelo/Business (Negocio): a las clases que se encuentran dentro de la capa de negocio después del nombre se le agrega la palabra “Model”. Ejemplo: DatDispositivoModel.

Clases del dominio (Domain): el nombre que reciben las clases que se encuentran dentro de la capa de dominio es el de la tabla en la base de datos. Ejemplo: DatEstacion.

Generated: el nombre de las clases que se encuentran dentro de *generated* comienza con el prefijo “Base” y continúa con el nombre de la tabla en la base de datos. Ejemplo: BaseDatEstacion.

Nomenclatura de las funciones

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido el sufijo “Action”. Ejemplo: modificarSellosAction.

Nomenclatura de las variables

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing. Ejemplo: $datos.

Normas de comentariado

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenimiento a lo largo del tiempo.

Estilo del código

En la implementación, al escribir las sentencias en php, se utilizará el tabulado del lenguaje como se muestran a continuación.

```php
<?php
        //código
?>
```

Sangría o indexado: La política de sangría a utilizar en la implementación es por tabulado. Ejemplo:

```php
<?php
    public function GenerarFichaTecnicaEstacionAction() {
        $idlocal = $this->_request->getPost('idlocal');
        $idestacion = $this->_request->getPost('id_estaciones')
        $pdf = new ExpedientePdf()
        $pdf->Open();
        $pdf->GenerarFichaEstacion($idlocal, $idestacion);
    }
?>
```

Brazas o llaves: En la declaración de clases o interfaces, métodos, bloques y switch, la apertura de llaves se hace en la misma línea. Ejemplo:

```php
<?php
    function importarDispositivosAction() {
        $idficha = $this->_request->getPost('idficha');
        $idestacion = $this->_request->getPost('idestacion');
        $dir_file = $_FILES['fileUpload']['tmp_name'];
        $model = new DatEstacionModel();
        if (file_exists($dir_file)) {
            $xml_cargado = simplexml_load_file($dir_file);
            $flag = $model->obtenerMAC($xml_cargado, $idestacion);
            if ($flag == 1) {
                $model->importar($xml_cargado, $idficha);
                $this->showMessage('Se han importado los dispositivos
                                    satisfactoriamente.');
            }
        }
        else{
            $this->showMessage('Los dispositivos seleccionados no
                                pertenecen a esta estación de trabajo.');
        }
    }
?>
```

3.3 HERRAMIENTA GETET

Una vez definidos los estándares de codificación a utilizar y establecidos todos los artefactos necesarios en la fase de diseño se procede a la implementación de GETET. A continuación se reflejan algunas interfaces de la herramienta relacionadas a cada uno de los tres módulos implementados así como la descripción de cada una de ellas.

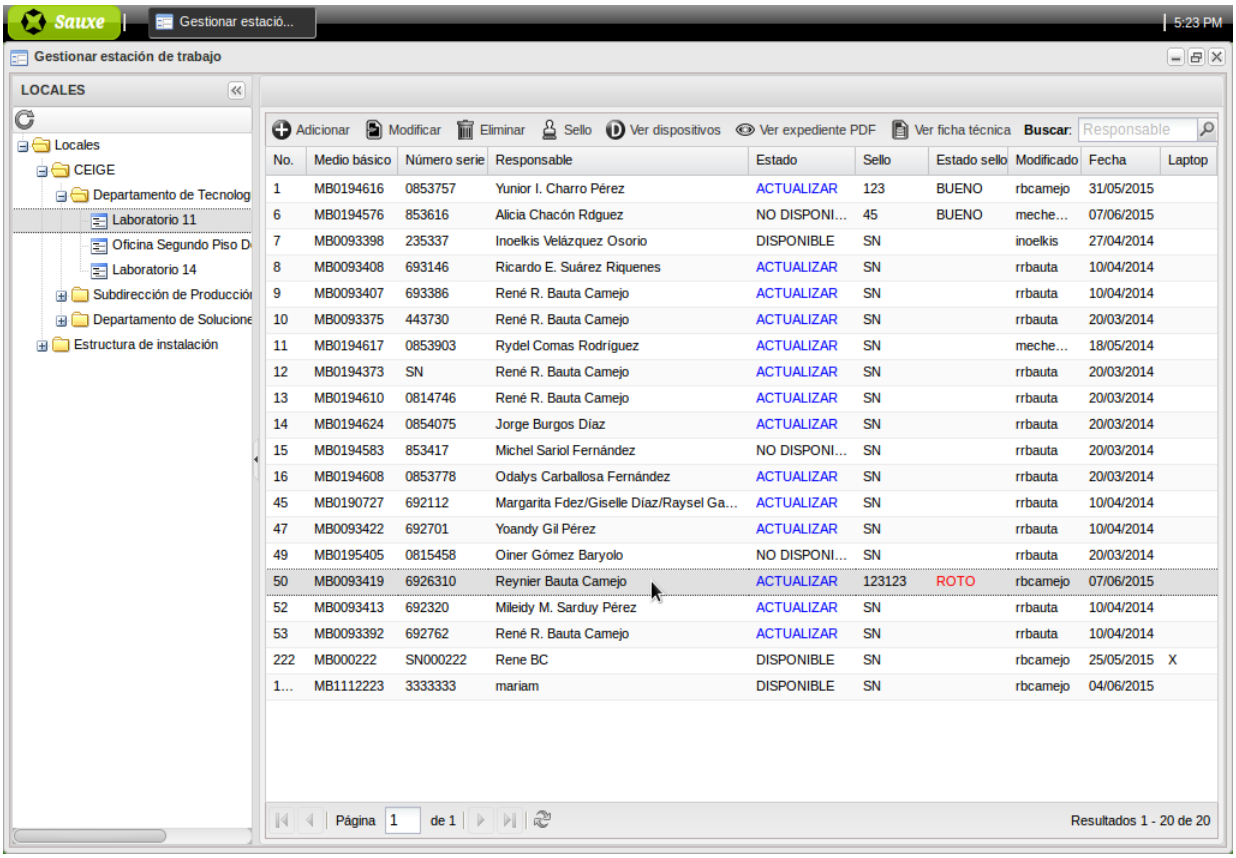


Figura 18: Interfaz de GETET correspondiente a la funcionalidad gestionar estación de trabajo.

La figura muestra la interfaz principal de GETET correspondiente a la funcionalidad gestionar estación de trabajo, que se encuentra dentro del módulo Gestionar Expediente. En esta interfaz se agrupan la mayoría de las funcionalidades por ejemplo: mostrar los dispositivos de una estación de trabajo, la generación de expedientes técnicos en formato PDF, la realización de auditorías y la muestra de los resultados de las mismas, entre otras funciones que se pueden observar en la imagen.

En la parte izquierda aparece definida, haciendo uso de un árbol, la estructura del CEIGE de acuerdo a sus diferentes áreas. En el centro se listan las estaciones de trabajo de acuerdo al local que se selecciona. Además permite realizar el cambio de local de una o más estaciones de trabajo con solo arrastrarlas sobre uno de los locales predefinidos en la parte izquierda.

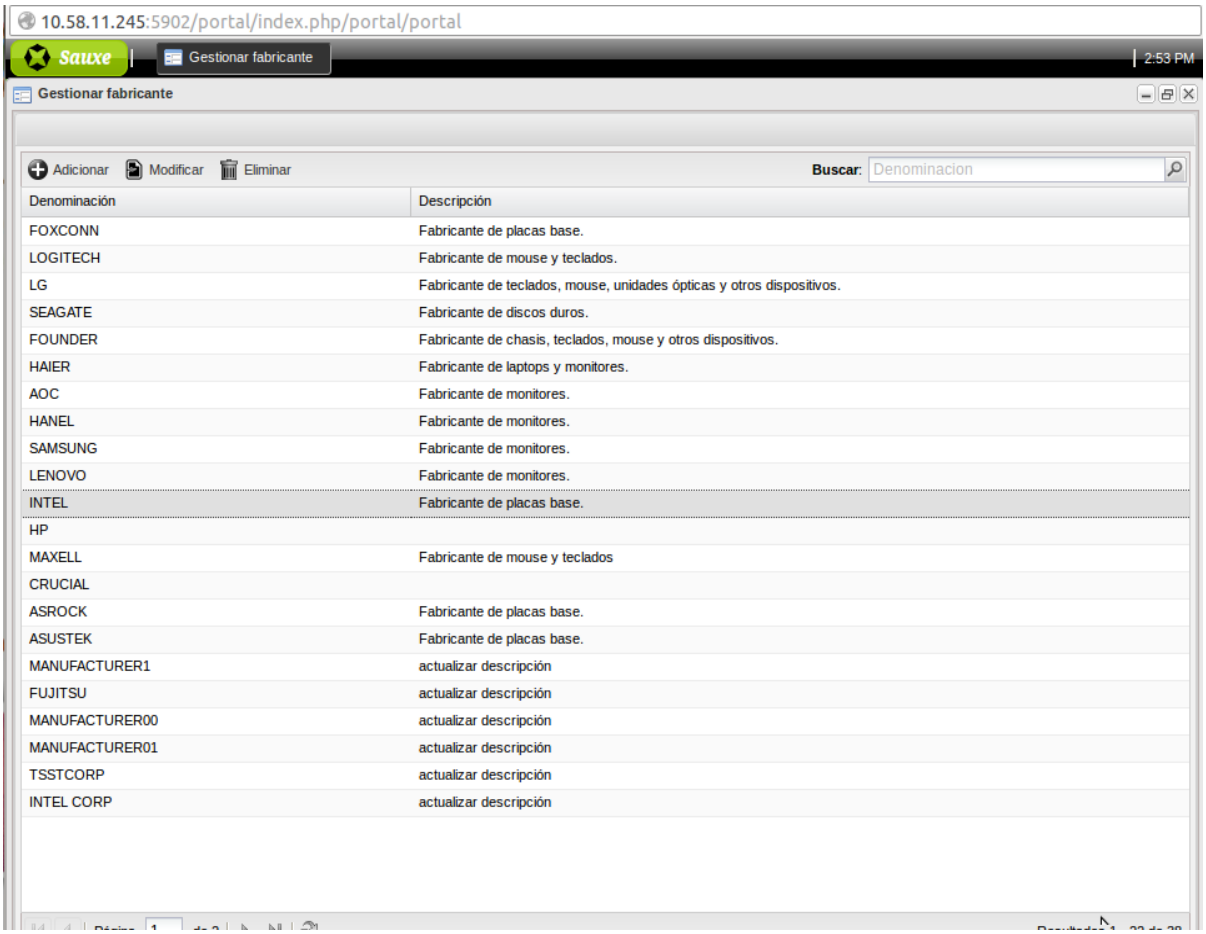


Figura 19: Interfaz de GETET correspondiente a la funcionalidad gestionar fabricante.

La Figura 19 muestra la interfaz de GETET correspondiente a la funcionalidad gestionar fabricante que se encuentra dentro del módulo Gestionar Nomencladores. Desde esta interfaz se pueden listar todos los fabricantes de dispositivos que han sido registrados en la aplicación y además existe la posibilidad de adicionar, modificar o eliminar estos dispositivos. También brinda la opción de realizar una búsqueda a través de la denominación del fabricante.

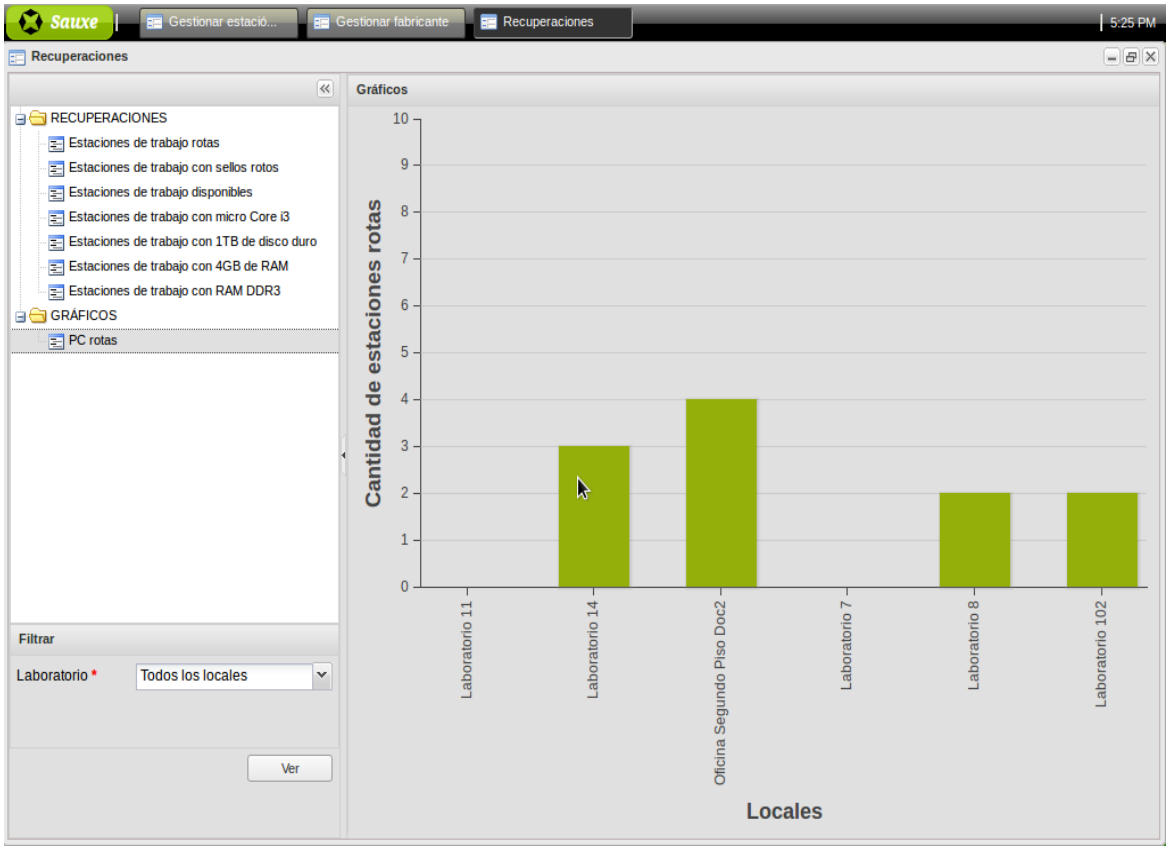


Figura 20: Interfaz de GETET correspondiente a la funcionalidad recuperaciones.

En la Figura 20 se muestra la interfaz para realizar las recuperaciones, funcionalidad que se encuentra dentro del módulo Reportes. En la parte izquierda aparecen predefinidos los reportes que se obtendrán en formato PDF, así como los gráficos que se pueden generar haciendo uso del filtro ubicado en la parte inferior.

3.4 MÉTRICAS DE SOFTWARE

Las métricas de *software* constituyen una medida cuantitativa que permite a los desarrolladores tener una visión profunda de la eficacia del proceso de *software*. En esta evaluación se reúnen los datos básicos referentes a los factores de calidad que posteriormente serán analizados, comparados con promedios anteriores y evaluados, para determinar las mejorías en términos de éstos factores. Pressman señala que las métricas serán utilizadas para la evaluación de determinados atributos de calidad. A continuación se exponen varios de estos atributos según los criterios de (Pressman, 2005).

Complejidad de implementación: consiste en el grado de dificultad que tiene implementar un determinado diseño de clases.

Responsabilidad: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.

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, está muy ligada a la característica de reutilización.

Complejidad del mantenimiento: consiste en el grado de esfuerzo necesario a realizar para realizar un arreglo, una mejora o una rectificación de algún error en el diseño del *software*. Puede influir indirectamente, pero fuertemente en los costes y la planificación del proyecto.

Cantidad de pruebas: asociado al número o al grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, entre otras) diseñado.

Tamaño operacional de clases (TOC)

Esta dado por el número de métodos asignados a una clase (Baryolo Gómez, 2010). La relación entre esta métrica y los atributos de calidad mencionados, se observan en la siguiente tabla.

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Tabla 3: Relación entre el TOC y los atributos de calidad.

Relación entre clases (RC)

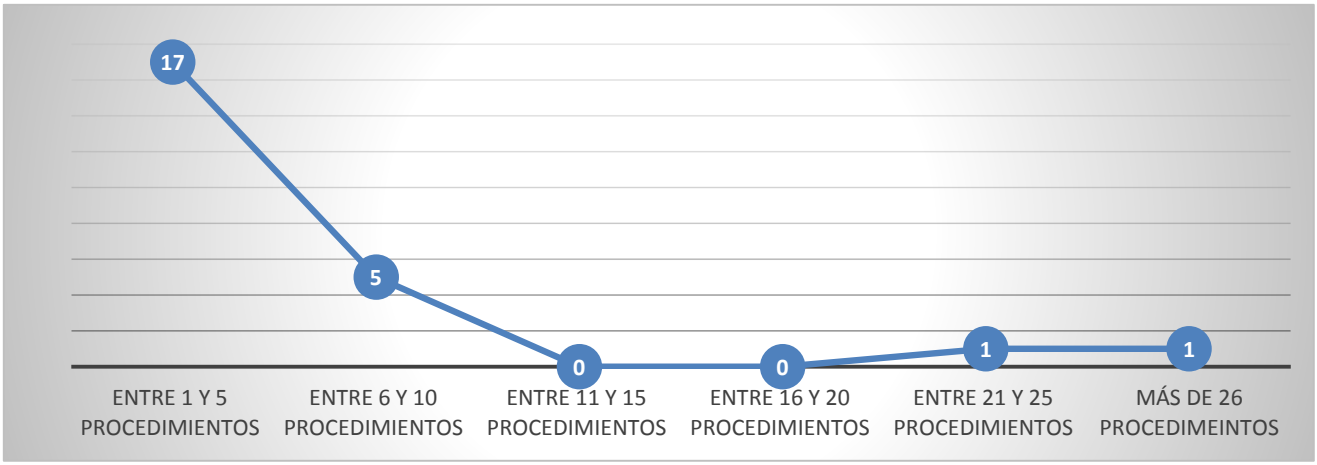
Está dado por el número de relaciones de uso de una clase con otra (Baryolo Gómez, 2010). La relación entre esta métrica y los atributos de calidad mencionados, se observa en la Tabla 4.

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad de mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 4: Relación entre RC y los atributos de calidad.

3.4.1 APLICACIÓN DE LA MÉTRICA TOC AL DISEÑO

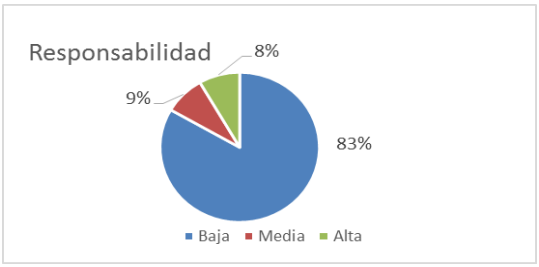
La Gráfica 1 muestra los resultados obtenidos de la aplicación de la métrica TOC al diseño de GETET. Estos resultados se agrupan en los intervalos representados en la gráfica. El gráfico refleja que la mayoría de las clases (17) tienen de uno a cinco procedimientos.



Gráfica 1: Representación de la evaluación de la métrica TOC.

Este resultado demuestra que el funcionamiento general del sistema está distribuido equitativamente entre la mayoría de las clases, aunque es necesario analizar las que contienen la mayor cantidad de procedimientos para tratar de restarle funcionalidades y distribuirlas entre las demás clases para evitar que alguna sea demasiado crítica y que en caso de fallo sea menor el número de funcionalidades que queden fuera de servicios.

En la siguiente gráfica se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad. La gráfica muestra un resultado satisfactorio ya que el 83 % de las clases poseen una baja responsabilidad.



Gráfica 2: Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.

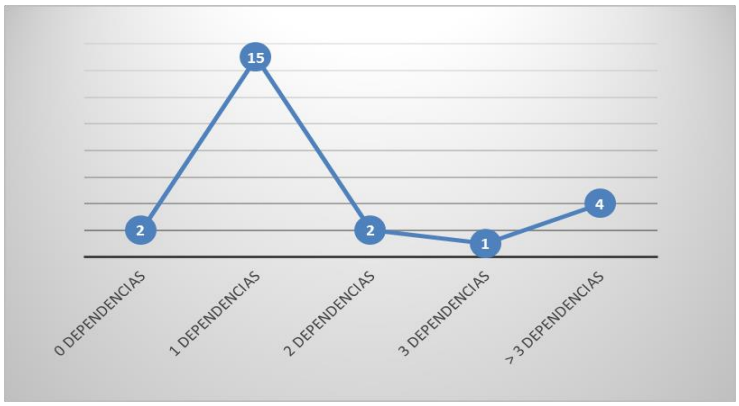
A continuación se muestra el resultado de la incidencia de la métrica TOC en el atributo complejidad de implementación (izquierda). Este gráfico muestra un resultado satisfactorio, pues el 83 % de las clases poseen una baja complejidad de implementación, característica que permite mejorar el mantenimiento y soporte de estas clases. Además la figura muestra en su parte derecha el resultado de la incidencia de la métrica TOC en el atributo reutilización. Este gráfico muestra que el diseño de la herramienta tiene un grado aceptable pues el 83 % de las clases posee una alta reutilización.

Gráfica 3: Resultados de la evaluación de la métrica TOC para el atributo complejidad y reutilización.

Teniendo en cuenta los resultados de la aplicación de la métrica TOC al diseño, se puede señalar que en general el diseño tiene una calidad aceptable, pues el mayor por ciento del total de las clases posee baja responsabilidad, baja complejidad de implementación y una alta reutilización. Los instrumentos y las tablas de resultados del instrumento de medición de la métrica TOC se pueden observar en el Anexo 2.

3.4.2 RESULTADOS DE LA APLICACIÓN DE LA MÉTRICA RC

En la siguiente figura se muestra los resultados obtenidos en el instrumento de evaluación de la métrica RC. El gráfico refleja que la mayoría de las clases (17) tienen entre 1 y ninguna dependencia con otra clase, demostrando que casi el total de las clases presentan niveles aceptables de calidad.

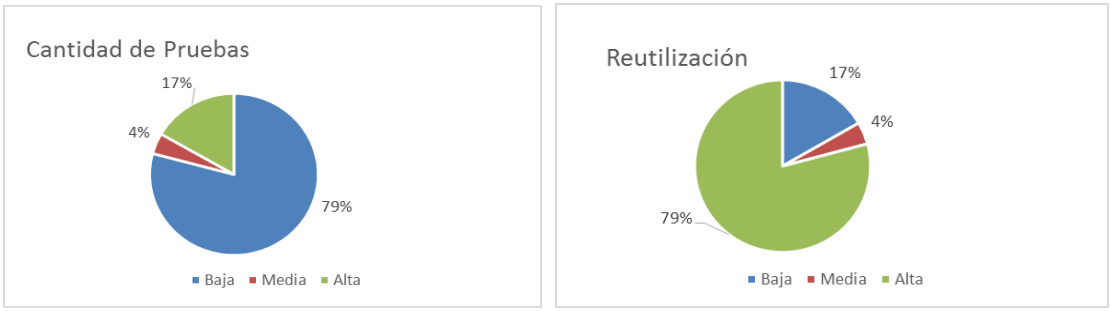


Gráfica 4: Representación de la evaluación de la métrica RC.

Las siguientes gráficas representan la incidencia de los resultados de la evaluación de la métrica RC en los atributos: acoplamiento (izquierda) y complejidad de mantenimiento (derecha). En el gráfico de la izquierda se evidencia un diseño eficiente al quedar reflejado que el 63 % de las clases presentan bajo acoplamiento, mientras que el 8 % no presentan dependencias con otras clases, lo que aumenta el grado de reutilización del componente. En el gráfico de la derecha el resultado es favorable ya que el 79 % de las clases presentan baja complejidad de mantenimiento, dándole respaldo a una buena calidad de la arquitectura.

Gráfica 5: Resultados de la evaluación de la métrica RC para el atributo acoplamiento y complejidad de mantenimiento.

A continuación se muestra la representación de la incidencia de la métrica RC en los atributos: cantidad de pruebas (izquierda) y reutilización (derecha). El gráfico de la izquierda demuestra que la mayor cantidad de las clases no necesita una cantidad elevada de pruebas según las dependencias entre las clases. Por su parte el gráfico de la derecha muestra que el diseño tiene un 79 % de reutilización de sus clases, constituyendo un factor importante en el desarrollo del *software*.



Gráfica 6: Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas y reutilización.

Analizando los resultados de la evaluación del instrumento de medición de la métrica relación entre clases (RC), se puede concluir que el diseño posee una calidad aceptable, ya que el 71 % de las clases de la solución tienen menos de 2 dependencias con otras clases. También se refleja en los resultados que en el 71 % el acoplamiento entre clases es mínimo; que la complejidad de mantenimiento y la cantidad de pruebas son bajas en un 79 %; y que la reutilización se comporta en 79%. Este resultado demuestra que los atributos de calidad se encuentran en niveles satisfactorios. Los instrumentos y las tablas de resultados del instrumento de medición de la métrica RC se pueden observar en el Anexo 3.

3.5 PRUEBAS DE SOFTWARE

3.5.1 RESULTADOS DE LA APLICACIÓN DE LAS PRUEBAS DE CAJA BLANCA

La técnica utilizada para realizar las pruebas de caja blanca a la Herramienta GETET es la del camino básico. A continuación se muestra el grafo generado durante la aplicación de esta técnica al código de la funcionalidad **cargarLocalesAction()** perteneciente a la clase **GstestacionController.php**. El fragmento del código utilizado se encuentra en el Anexo 4.

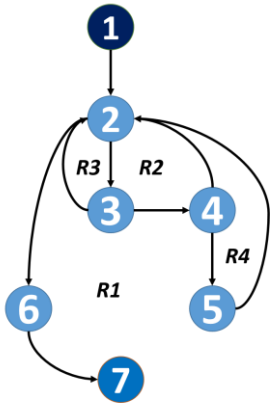


Figura 21: Grafo del camino básico asociado a la funcionalidad cargarLocalesAction()

La complejidad ciclomática es la métrica de *software* que proporciona una medición cuantitativa de la complejidad lógica de un programa. Esta métrica calcula la cantidad de caminos independientes de cada una de las funcionalidades del programa. También provee el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. El cálculo de la complejidad ciclomática se realiza después de construir el grafo y para ello se utilizan tres fórmulas que aseguran que esta métrica se ha calculado correctamente si coinciden sus resultados (Pressman, 2005).

Fórmula 1: V(G) = R donde R representa la cantidad total de regiones.

$$V(G) = 4$$

Fórmula 2: V(G) = A–N+2 donde A es el número de aristas del grafo de flujo y N es el número de nodos.

$$V(G)=9–7+2$$
$$V(G)=4$$

Fórmula 3: V(G) = P+1 donde P es el número de nodos predicado contenidos en el grafo de flujo (se denomina nodo predicado a los nodos de los cuales parten dos o más aristas).

$$V(G)=3+1$$
$$V(G)=4$$

Posterior al cálculo de las fórmulas mencionadas, se pudo constatar que las tres dan el mismo resultado. Por tanto se puede afirmar que la complejidad ciclomática del método cargarLocalesAction() tiene un valor de 4. Este valor significa que existen 4 caminos posibles para el flujo del método:

Camino básico #1: 1, 2, 3, 4, 5, 2, 6,7
Camino básico #2: 1, 2, 3, 2, 6,7
Camino básico #3: 1, 2, 3, 4, 2, 6,7
Camino básico #4: 1, 2, 6, 7

Además, este valor representa la mínima cantidad de casos de prueba para el procedimiento, teniendo en cuenta que se realiza un caso de prueba por cada camino básico:

Caso de prueba para el camino básico #1: Si count($locales)>0, Si $local->idestructuraop != null, Si $local->idpadre != $local->idestructuraop
Caso de prueba para el camino básico #2: Si count($locales)>0, Si $local->idestructuraop == null
Caso de prueba para el camino básico #3: Si count($locales)>0, Si $local->idestructuraop != null, Si $local->idpadre == $local->idestructuraop
Caso de prueba para el camino básico #4: Si count($locales) == 0

Con la realización de estas pruebas se garantizó que todas las sentencias del programa fueran ejecutadas al menos una vez.

3.5.2 RESULTADOS DE LA APLICACIÓN DE LAS PRUEBAS FUNCIONALES O DE CAJA NEGRA

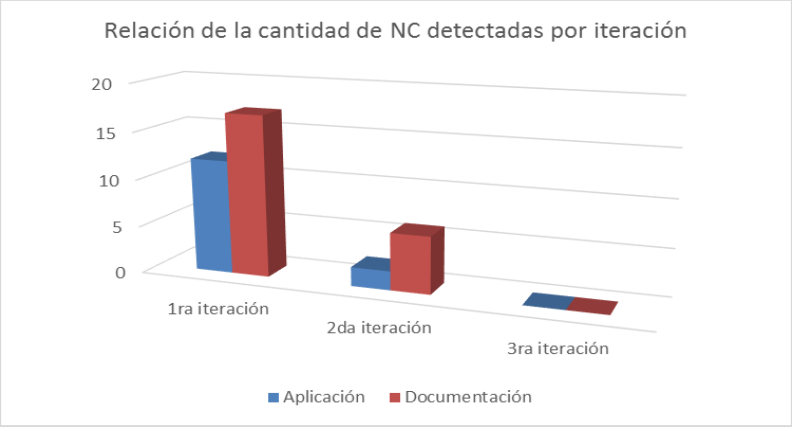
Para evaluar los requisitos funcionales de GETET en el CEIGE se realizaron pruebas de caja negra en la fase de pruebas internas, utilizando la técnica de partición equivalente. Para aplicarla se confeccionaron los diseños de casos de prueba para cada una de las funcionalidades del sistema los cuales pueden ser consultados en el expediente de proyecto. Para comprobar la calidad de la herramienta se realizaron 3 revisiones por el departamento Desarrollo de Componentes del centro. Las no conformidades (NC) encontradas se clasificaron en: detectadas en la aplicación y detectadas en la documentación.

En la siguiente tabla se muestra la cantidad de no conformidades detectadas en la aplicación y la documentación por cada iteración. De igual forma en la Gráfica 7 se muestran estos resultados de

forma gráfica para facilitar el análisis. Se puede observar como en la primera iteración se detectan un total de 29 NC que son corregidas en su totalidad en una tercera iteración.

Tipo de NC	1ra iteración	2da iteración	3ra iteración
Aplicación	12	2	0
Documentación	17	6	0

Tabla 5: Relación de las NC detectadas.



Gráfica 7: Relación de las NC detectadas.

3.6 VALIDACIÓN DE LA HERRAMIENTA GETET

Para la validación de la herramienta se tuvieron en cuenta dos indicadores: el alcance de los requisitos y la aceptación del cliente.

Alcance de los requisitos

Del total de 43 requisitos de *software* identificados se logró implementar el 100 %, dando cumplimiento a todas las necesidades de los clientes.

Aceptación del cliente

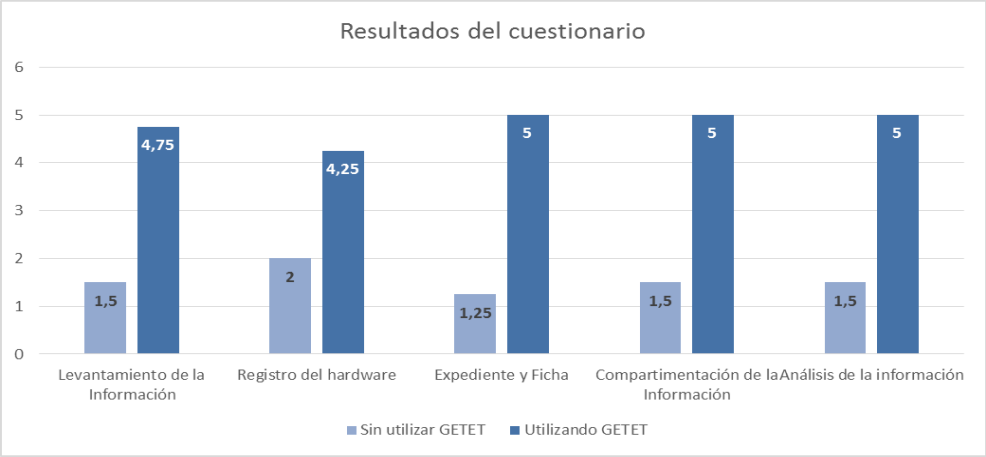
Con el objetivo de validar la propuesta se realizaron pruebas de aceptación por parte del cliente para determinar si la herramienta cumplía o no las necesidades que existían en relación a la gestión de expedientes técnicos de las estaciones de trabajo en el centro CEIGE. Para realizar lo anterior se creó una comisión en el proyecto Marco de Trabajo para el Desarrollo de Aplicaciones Web de

Gestión, la cual se encargó de realizar una revisión técnica de la herramienta. Esta comisión determinó que GETET estaba estable y lista para su uso. La comisión emitió el acta de liberación correspondiente, reflejada en el Anexo 5, evidenciando que la herramienta cumple con todas las expectativas y necesidades del CEIGE. Asimismo la Dirección del Centro de Software Libre emitió un aval donde reflejaban que utilizando GETET crearon el 95% de los expedientes técnicos del área contribuyendo a la realización de un trabajo más eficaz y eficiente en el área de tecnología del centro. Este aval se puede consultar en el Anexo 6. También en el Anexo 7 se puede encontrar otro aval emitido por el Especialista de Seguridad Informática y Administración de la Configuración de CEIGE expresando su satisfacción con GETET.

3.7 VALIDACIÓN DE LA INVESTIGACIÓN

En el centro CEIGE el control de la información de la estaciones de trabajo se realizaba de forma manual haciendo uso de la herramienta Microsoft Excel, la cual no está exenta a que se introduzcan errores afectando el análisis de la información y la toma de decisiones. Asimismo los expedientes y fichas técnicas se elaboraban de forma manual dificultando la entrega y calidad de la información.

Con el objetivo de aportar una solución factible al problema que presentaba el centro se decide emplear una herramienta para la gestión de los expedientes técnicos de las estaciones de trabajo del CEIGE para aumentar el control sobre el *hardware* de las estaciones de trabajo. Para validar la investigación, se confeccionó un cuestionario para ser respondido por los jefes de departamento del centro, el asesor de tecnología del centro y el asesor de tecnología del Centro de Soluciones Libres. En el cuestionario se realizaron las preguntas con el objetivo de evaluar cada una en un rango de valores de 1 a 5 en cuanto a la veracidad de la afirmación, teniendo para 1 las menos ciertas y para 5 las más ciertas. El cuestionario puede ser consultado en el Anexo 8. A continuación se muestra una gráfica que refleja los resultados obtenidos luego de aplicar el cuestionario.



Gráfica 8: Relación de los resultados obtenido en el cuestionario.

Como se puede observar en la gráfica, el control sobre el *hardware* de las estaciones de trabajo en el CEIGE aumenta considerablemente; antes de utilizar GETET se obtiene un 31% para este indicador, sin embargo, luego de utilizar la herramienta propuesta por la presente investigación el control alcanza niveles de un 96%, evidenciando un aumento en un 65%. Este resultado permite validar la investigación y garantizar el cumplimiento de su objetivo general.

3.8 CONCLUSIONES DEL CAPÍTULO

Una vez finalizado el capítulo se puede arribar a las siguientes conclusiones:

- El diagrama de componentes permitió representar los aspectos físicos del sistema a desarrollar mediante una colección de arcos y nodos.
- La evaluación aplicada al diseño mediante las métricas TOC y RC, evidencia niveles satisfactorios como el 83% de baja responsabilidad y el 71 % de reutilización de las clases del sistema.
- Las pruebas estructurales realizadas al código a partir de la técnica del camino básico evidencian una correcta implementación, garantizando que todas las sentencias del programa se ejecutan al menos una vez. Los resultados de las pruebas funcionales, en las que con tres iteraciones totales se logró dar respuesta a las no conformidades detectadas, acreditan la correcta puesta en práctica de los requisitos funcionales.
- La aplicación del método pre-experimento arrojando resultados satisfactorios permitió validar la investigación realizada.

CONCLUSIONES GENERALES

Luego de finalizada la investigación, presentada la propuesta de solución y realizadas las pruebas de *software* se puede arribar a las siguientes conclusiones:

- El estudio de bibliografía actualizada y científica permitió sentar las bases para el desarrollo de la investigación propuesta e identificar posibles puntos de reutilización con los sistemas semejantes estudiados.
- La realización del análisis y diseño de la solución permitió iniciar la implementación contando con productos de trabajo que favorecen el proceso de desarrollo siendo estos la entrada fundamental para la disciplina Implementación.
- La implementación de una herramienta para la gestión de los expedientes técnicos de las estaciones de trabajo en el CEIGE facilitó el proceso de inventario de *hardware* en el centro antes mencionado favoreciendo el control sobre sus recursos.
- Las validaciones realizadas a los requisitos y al diseño, las pruebas de caja blanca y caja negra ejecutadas sobre el código y la interfaz de usuario así como la validación de la investigación evidencian la calidad durante el proceso de desarrollo y el cumplimiento del objetivo trazado.

RECOMENDACIONES

Continúa siendo recomendación:

- Mejorar el módulo Recuperaciones para darle la oportunidad al usuario de escoger el tipo de reporte que desee realizar.
- Aumentar el número de reportes gráficos para facilitar el entendimiento de la información obtenida en los mismos.
- Realizar la captura de la información relacionada con el *hardware* de las estaciones de trabajo a través de la red.

REFERENCIAS BIBLIOGRÁFICAS

ACUÑA BRITO, K. eumed.net. eumed.net. In., 2013.

ACUÑA, K. B. Selección de metodologías de desarrollo para aplicaciones web. In., 2009.

ALTOVA. Diagramas de secuencia UML. ALTOVA. In., 2014.

BARYOLO GÓMEZ, O. Solución informática de autorización en entornos multientidad y multisistema. 2010.

BUSCHMANN, F., H. KEVLINN AND D. C. SCHMIDT. Pattern-Oriented Software Architecture: A pattern language for distribuited computing. 2007. ISSN 978-0-470-05902-9.

BUSCHMANN, F., R. MEUNIER, H. ROHNERT, P. SOMMERLAD, et al. Pattern–Oriented Software Architecture. A System of Patterns. 1996.

CAMACHO, E., F. CARDESO AND G. NUÑEZ. Arquitecturas de software. Guía de estudio. 2004.

CARÚS LLERA, A. AND J. SÁNCHEZ SANTANA. Módulo de Gestión de Inventarios e Incidencias de Hardware en Windows. Universidad de las Ciencias Informáticas, 2012.

COBO ROMANÍ, J. C. El concepto de tecnologías de la información. Benchmarking sobre las definiciones de las TIC en la sociedad del conocimiento. 2009, vol. 14, pp. 295-318. ISSN 1137-1102.

DATE, C. J. Introducción a los sistemas de bases de datos. 2001. ISSN 968-444-419-2.

DNA, N. S. Inventario de hardware y software - Gestión de equipos. 2012.

DOCTRINE. Doctrine. Doctrine-Project. In., 2014.

ECMA. JavaScript Introduction. In.: W3Schools.com, 2010.

ENTEL. UML: Un Lenguaje Modelo. In CIENTEC. Grupo ENTEL, 2013.

GRAU, R., C. CORREA AND M. ROJAS. Metodología de la investigación. Segunda edición. 2004.

GUNDA SAI, G. Requirements Engineering: Elicitation Techniques. 2008.

GUTMANS, A., S. BAKKEN STIG AND R. DERICK. PHP 5 Power Programing. In.: Professional Technical Reference, 2004.

HAMON, H. 2014. Identifying Design Patterns in the Symfony Framework. In Proceedings of the Symfony Conference, Istanbul, Turkey, 2/5/2014 2014.

HERNÁNDEZ PÉREZ, J. A., Y. ORDOÑES LEYVA AND E. AVILES VAZQUEZ. GESTIÓN DE INCIDENCIAS EN INVENTARIOS DE RED. 2013a.

HERNÁNDEZ PÉREZ, J. A., Y. ORDOÑES LEYVA, S. RAMOS BETANCOURT AND I. LEÓN RODRÍGUEZ. MÓDULO DE GESTIÓN DE INCIDENCIAS Y ACCIONES DE LA PLATAFORMA DE GESTIÓN DE RECURSOS DE INVENTARIOS DE HARDWARE Y SOFWARE [online]. Informática 2013, 2013b.

IEEE. Introducción a la Ingeniería de Requisitos. Ingeniería de software. 2007.

JACOBSON, I., G. BOOCH AND J. RUMBAUGH. El Lenguaje Unificado de Modelado. 1999.

LARMAN, C. UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado [online]. [México]: 1999.

LEÓN, R. A. H. AND Z. C. GONZÁLEZ. El paradigma Cuantitativo de la Investigación Científica. 2008. ISSN 978-959-16-9343-2.

LOBO, A. R. ComponentBuilder. Manual del Desarrollador. . 2012.

LÓPEZ, J. M. Cliente para Subversion para principiantes y expertos 2010.

MARK. Learning Python, Fourth Edition. 2010.

MARQUINA, E. P., JOSE DAVID Guía de Patrones, Prácticas y Arquitectura .NET 2008.

MODX. OCS inventory next generation, Description of all features about OCS Inventory NG. 2011a.

MODX. OCS Inventory NG. OCS Inventory NG. In., 2011b.

NETSUPPORT. NetSupport DNA. 2014. Available from Internet:<http://www.netsupportdna.com/>.

ORACLE, C. FOSS License Exception. MySql 2012.

ORDOÑES L, Y., E. AVILÉS V, J. HERNÁNDEZ P AND O. RODRÍGUEZ H. GRHS: Gestor de Recursos de Hardware y Software. 2014, pp. 5.

PÉREZ ALFONSO, D. Normas y estándares de codificación de Sauxe. 2012.

PGADMIN. Excelente gestor PostGreSQL. Puro Software. In., 2011.

PRESSMAN, R. S. Ingenería del Software. Un enfoque práctico: Quinta Edición [online]. 2005.

PRIETO, F. Patrones de diseño. 2009.

RALE. Diccionario Ilustrado Océano de la Lengua Española. In MILANESAT. Edificio Océano Barcelona, España, 2001, p. 1192.

REYNOSO, B. Architect Academy: Seminario de Arquitectura de Software. 2004.

RODRÍGUEZ SÁNCHEZ, T. Metodología de desarrollo para la Actividad productiva de la UCI 2014.

SERRADILLA, J. L. Control de Versiones con Subversion y TortoiseSVN. In.: ATICA. Unicversidad de Murcia, 2008.

SHEA, F. AND C. RAMSAY. Learning Ext JS. Birmingham: Packt Publishing Ltd [online]. 2008.

SOMMERVILLE, I. Ingeniería de Software. 7ma edición. 2005.

SPERBERG, C. Sobre convenciones y notaciones 2012. Available from Internet:<http://blog.unreal4u.com/2011/03/sobre-convenciones-y- notaciones-hungara-camelcase-etc/.>.

SVENSK, M. DiVA 2012.

TEDESCHI, N. Qué es un Patrón de Diseño. In., 2012.

THEAPACHE. Apache httpd 2.0.65 Released. The Apache HTTP server Project. In., 2013.

UCI. Universidad de las Ciencias Informáticas. In. Cuba, 2015.

VEO. VEO Ultimate. In. México, 2014.

VIDAL, J. Fundamentos de gestión y control de inventarios. 2012.

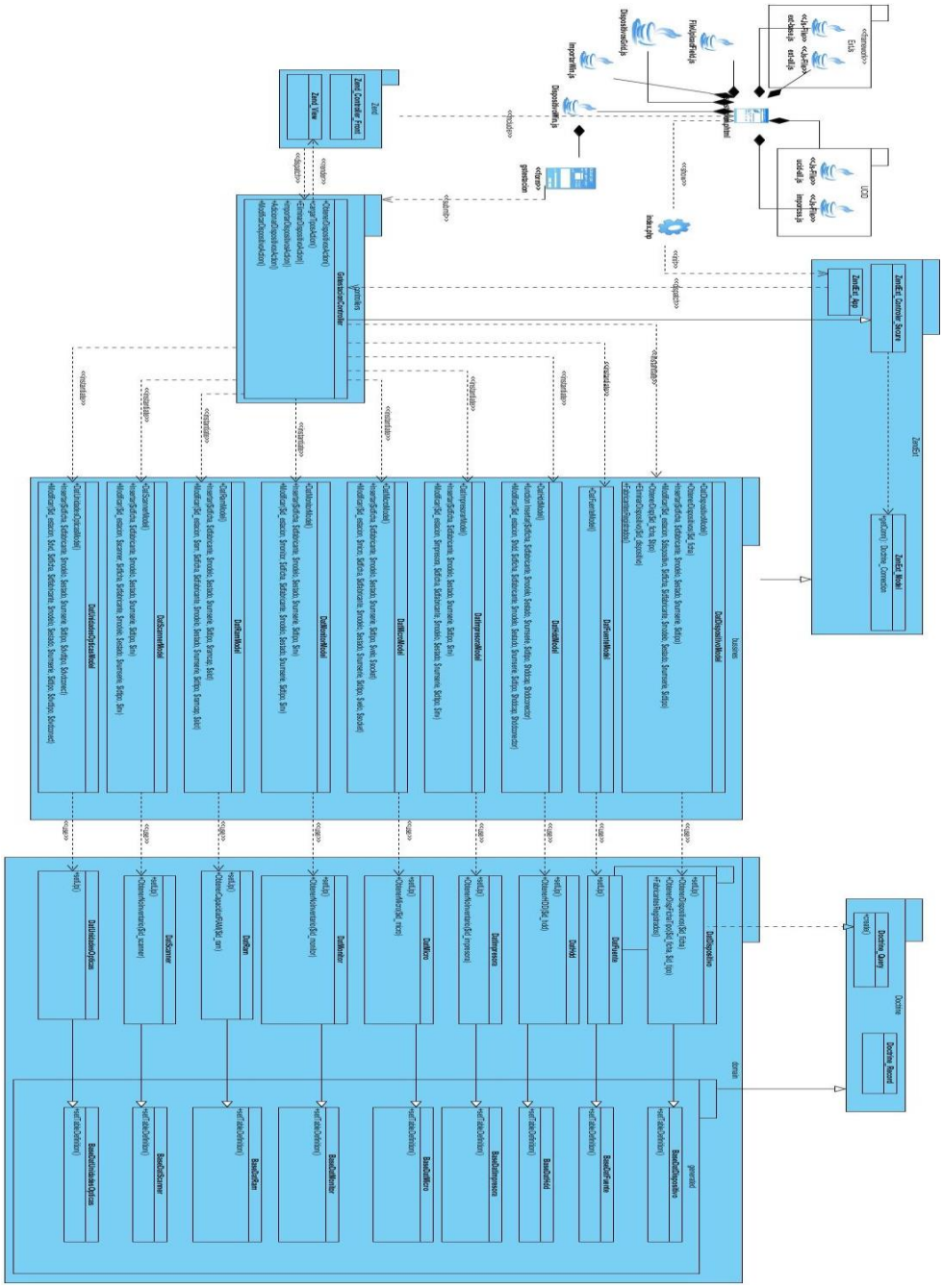
VISCONTI, M. AND H. ASTUDILLO. Fundamentos de Ingeniería de Software. In., 2006.

VISUAL-PARADIGM. UML, BPMN and Database Tool for Software Development. Visual Paradigm. In., 2013.

ZEND-TECHNOLOGIES. Zend Framework. In., 2013.

ANEXOS

Anexo 1: Diagrama de clase de Gestionar dispositivo asociado a las estaciones de trabajo.



Anexo 2: Instrumento de evaluación de las métrica TOC.

No	Subsistema	Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
1	Expediente técnico	GstestacionController	25	Alta	Alta	Baja
2	Expediente técnico	DatDispositivoModel	6	Baja	Baja	Alta
3	Expediente técnico	DatEstacionModel	30	Alta	Alta	Baja
4	Expediente técnico	DatExpedienteModel	2	Baja	Baja	Alta
5	Expediente técnico	DatFichaTecnicaModel	6	Baja	Baja	Alta
6	Expediente técnico	DatFuenteModel	0	Baja	Baja	Alta
7	Expediente técnico	DatHddModel	2	Baja	Baja	Alta
8	Expediente técnico	DatImpresoraModel	2	Baja	Baja	Alta
9	Expediente técnico	DatLocalModel	2	Baja	Baja	Alta
10	Expediente técnico	DatMicroModel	2	Baja	Baja	Alta
11	Expediente técnico	DatMonitorModel	2	Baja	Baja	Alta
12	Expediente técnico	DatRamModel	2	Baja	Baja	Alta
13	Expediente técnico	DatRegistroaudModel	2	Baja	Baja	Alta
14	Expediente técnico	DatReporteRoturaModel	1	Baja	Baja	Alta
15	Expediente técnico	DatResultadoaudModel	2	Baja	Baja	Alta
16	Expediente técnico	DatScannerModel	2	Baja	Baja	Alta
17	Expediente técnico	DatSelloModel	4	Baja	Baja	Alta
18	Expediente técnico	DatUnidadesOpticasModel	2	Baja	Baja	Alta
19	Expediente técnico	ExpedientePdf	8	Media	Media	Media
20	Expediente técnico	GstselladorController	4	Baja	Baja	Alta
21	Expediente técnico	NomSelladorModel	6	Baja	Baja	Alta
22	Expediente técnico	GsfabricanteController	4	Baja	Baja	Alta
23	Expediente técnico	NomFabricanteModel	7	Media	Media	Media
24	Expediente técnico	GsttipoController	4	Baja	Baja	Alta
25	Expediente técnico	NomTipoModel	7	Media	Media	Media
26	Expediente técnico	GstreportrotController	4	Baja	Baja	Alta
27	Expediente técnico	DatReporteRoturaModel	4	Baja	Baja	Alta
28	Expediente técnico	RecupController	9	Media	Media	Media

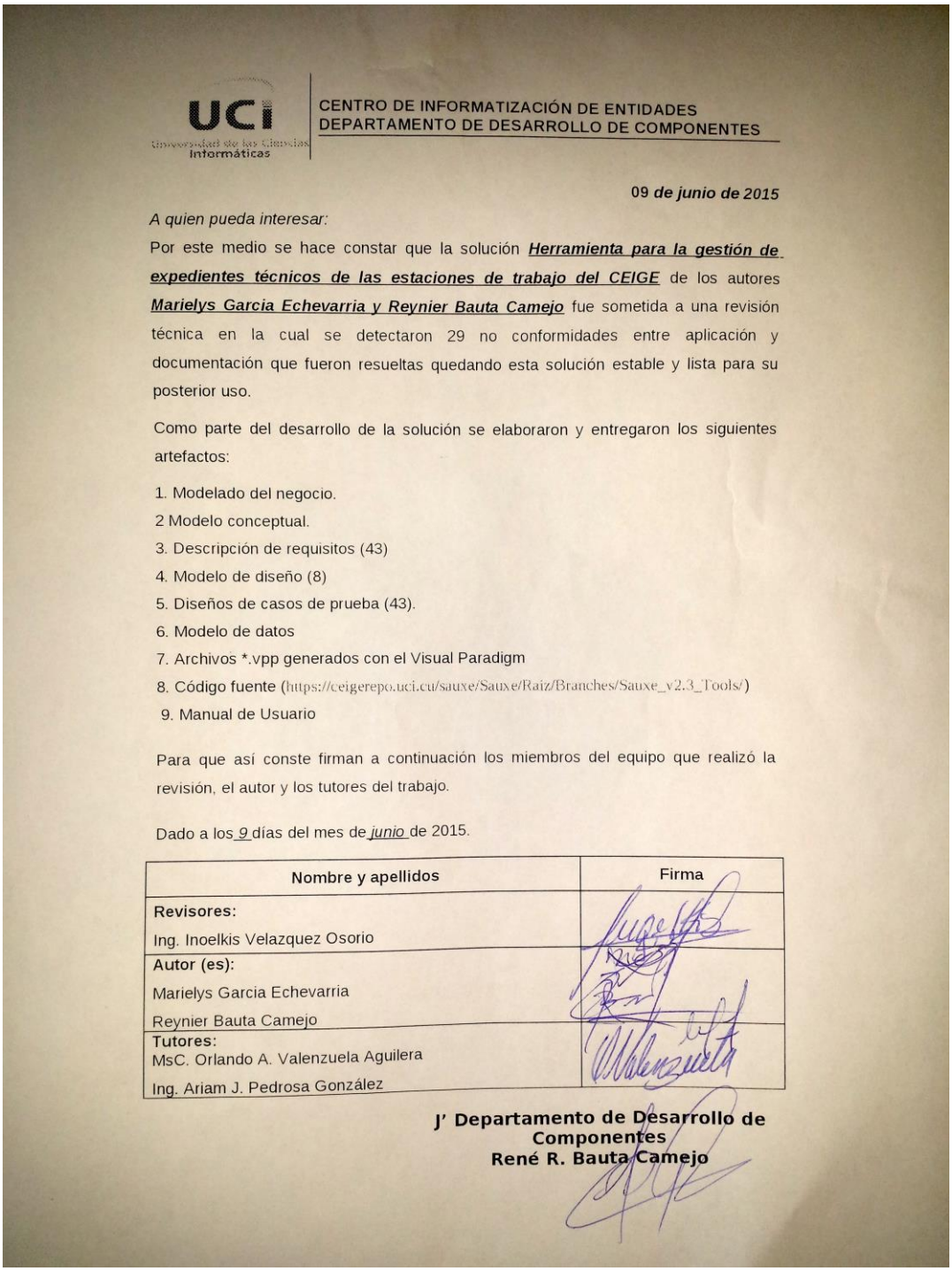
Anexo 3: Instrumento de evaluación de las métrica RC.

No	Subsistema	Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
1	Expediente técnico	GstestacionController	20	Alto	Alta	Baja	Alta
2	Expediente técnico	DatDispositivoModel	10	Alto	Alta	Baja	Alta
3	Expediente técnico	DatEstacionModel	9	Alto	Alta	Baja	Alta
4	Expediente técnico	DatExpedienteModel	1	Bajo	Baja	Alta	Baja
5	Expediente técnico	DatFichaTecnicaModel	3	Alto	Media	Media	Media
6	Expediente técnico	DatFuenteModel	0	Ninguno	Baja	Alta	Baja
7	Expediente técnico	DatHddModel	1	Bajo	Baja	Alta	Baja
8	Expediente técnico	DatImpresoraModel	1	Bajo	Baja	Alta	Baja
9	Expediente técnico	DatLocalModel	1	Bajo	Baja	Alta	Baja
10	Expediente técnico	DatMicroModel	1	Bajo	Baja	Alta	Baja
11	Expediente técnico	DatMonitorModel	1	Bajo	Baja	Alta	Baja
12	Expediente técnico	DatRamModel	1	Bajo	Baja	Alta	Baja
13	Expediente técnico	DatRegistroaudModel	1	Bajo	Baja	Alta	Baja
14	Expediente técnico	DatReporteRoturaModel	0	Ninguno	Baja	Alta	Baja
15	Expediente técnico	DatResultadoaudModel	2	Medio	Baja	Alta	Baja
16	Expediente técnico	DatScannerModel	1	Bajo	Baja	Alta	Baja
17	Expediente técnico	DatSelloModel	1	Bajo	Baja	Alta	Baja
18	Expediente técnico	DatUnidadesOpticasModel	1	Bajo	Baja	Alta	Baja
19	Expediente técnico	ExpedientePdf	12	Alto	Alta	Baja	Alta
20	Expediente técnico	GstselladorController	2	Medio	Baja	Alta	Baja
21	Expediente técnico	NomSelladorModel	1	Bajo	Baja	Alta	Baja
22	Expediente técnico	GsfabricanteController	1	Bajo	Baja	Alta	Baja
23	Expediente técnico	NomFabricanteModel	1	Bajo	Baja	Alta	Baja
24	Expediente técnico	GsttipoController	1	Bajo	Baja	Alta	Baja
25	Expediente técnico	NomTipoModel	1	Bajo	Baja	Alta	Baja
26	Expediente técnico	GstreportrotController	3	Alto	Media	Media	Media
27	Expediente técnico	DatReporteRoturaModel	2	Medio	Baja	Alta	Baja
28	Expediente técnico	RecupController	1	Bajo	Baja	Alta	Baja

Anexo 4: Código fuente utilizado para realizar las pruebas de caja blanca.

```php
<?php
public function cargarLocalesAction(){
        $locales = $this->integrator->metadatos->ListadoEstructurasT();                    //1
        $cont = 1;                                                                         //1
        $estNoformales = array();                                                          //1
        $estNoformales[0]['idestructuraop'] = 1;                                           //1
        $estNoformales[0]['denominacion'] = 'Todos los locales';                           //1
        foreach ($locales as $local){                                                      //2
            if ($local->idestructuraop && $local->idpadre != $local->idestructuraop){     //3
                $estNoformales[$cont]['idestructuraop'] = $local->idestructuraop;          //4
                $estNoformales[$cont]['denominacion'] = $local->denominacion;              //5
                $cont++;
            }
        }                                                                                  //6
        echo json_encode($estNoformales);                                                  //7
    }
?>
```

Anexo 5: Acta de liberación del proyecto Marco de Trabajo para el Desarrollo de Aplicaciones Web de Gestión.



UCI Universidad de las Ciencias Informáticas

CENTRO DE INFORMATIZACIÓN DE ENTIDADES
DEPARTAMENTO DE DESARROLLO DE COMPONENTES

09 *de junio de 2015*

A quien pueda interesar:

Por este medio se hace constar que la solución ***Herramienta para la gestión de expedientes técnicos de las estaciones de trabajo del CEIGE*** de los autores ***Marielys Garcia Echevarria y Reynier Bauta Camejo*** fue sometida a una revisión técnica en la cual se detectaron 29 no conformidades entre aplicación y documentación que fueron resueltas quedando esta solución estable y lista para su posterior uso.

Como parte del desarrollo de la solución se elaboraron y entregaron los siguientes artefactos:

1. Modelado del negocio.
2 Modelo conceptual.
3. Descripción de requisitos (43)
4. Modelo de diseño (8)
5. Diseños de casos de prueba (43).
6. Modelo de datos
7. Archivos *.vpp generados con el Visual Paradigm
8. Código fuente (https://ceigerepo.uci.cu/sauxe/Sauxe/Raiz/Branches/Sauxe_v2.3_Tools/)
9. Manual de Usuario

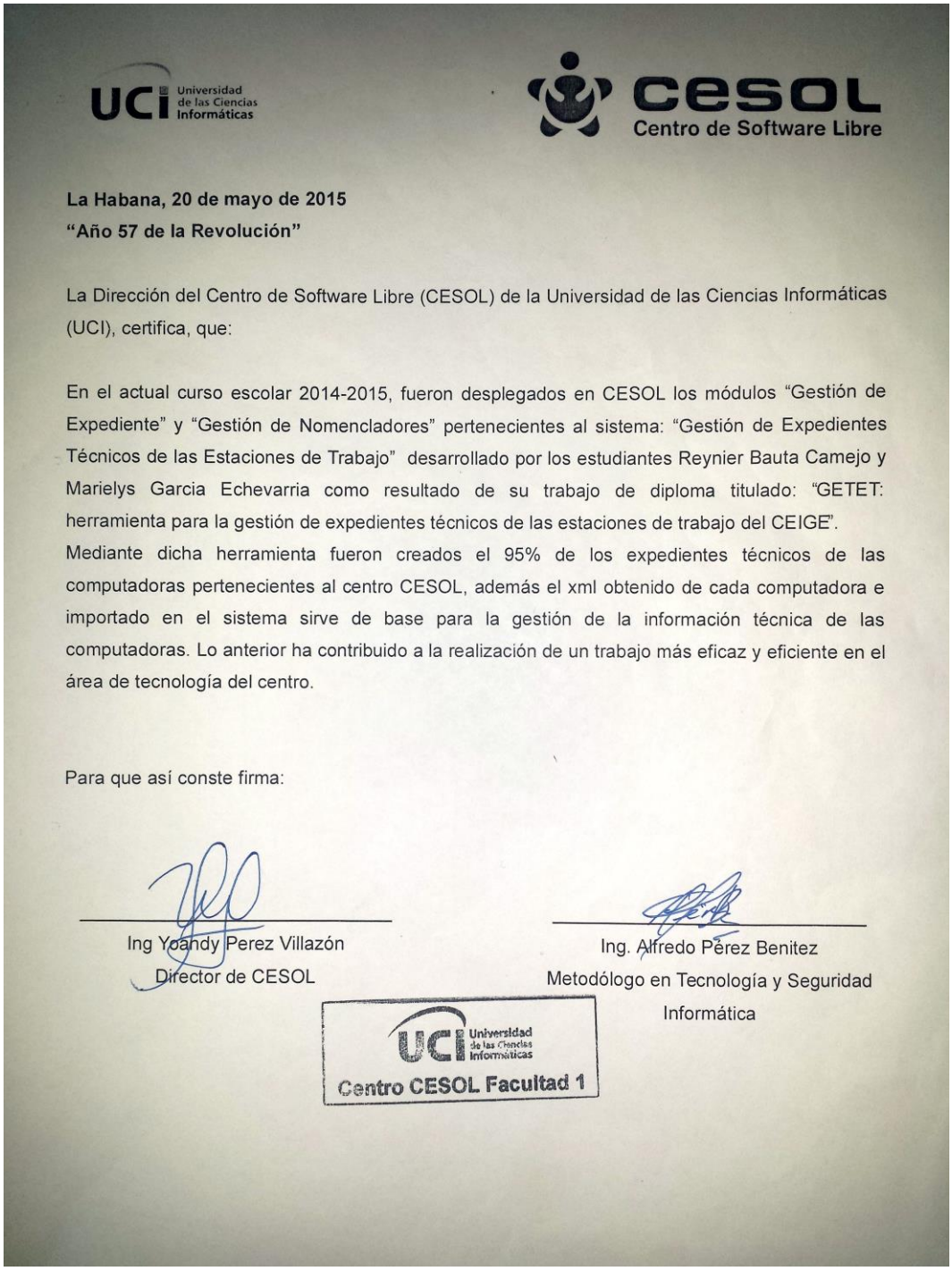
Para que así conste firman a continuación los miembros del equipo que realizó la revisión, el autor y los tutores del trabajo.

Dado a los *9* días del mes de *junio* de 2015.

Nombre y apellidos	Firma
Revisores: Ing. Inoelkis Velazquez Osorio	
Autor (es): Marielys Garcia Echevarria Reynier Bauta Camejo	
Tutores: MsC. Orlando A. Valenzuela Aguilera Ing. Ariam J. Pedrosa González	

J' Departamento de Desarrollo de Componentes
René R. Bauta Camejo

Anexo 6: Aval emitido por la Dirección del Centro de Software Libre CESOL.



UCI Universidad de las Ciencias Informáticas

CESOL Centro de Software Libre

La Habana, 20 de mayo de 2015
"Año 57 de la Revolución"

La Dirección del Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas (UCI), certifica, que:

En el actual curso escolar 2014-2015, fueron desplegados en CESOL los módulos "Gestión de Expediente" y "Gestión de Nomencladores" pertenecientes al sistema: "Gestión de Expedientes Técnicos de las Estaciones de Trabajo" desarrollado por los estudiantes Reynier Bauta Camejo y Marielys Garcia Echevarria como resultado de su trabajo de diploma titulado: "GETET: herramienta para la gestión de expedientes técnicos de las estaciones de trabajo del CEIGE".
Mediante dicha herramienta fueron creados el 95% de los expedientes técnicos de las computadoras pertenecientes al centro CESOL, además el xml obtenido de cada computadora e importado en el sistema sirve de base para la gestión de la información técnica de las computadoras. Lo anterior ha contribuido a la realización de un trabajo más eficaz y eficiente en el área de tecnología del centro.

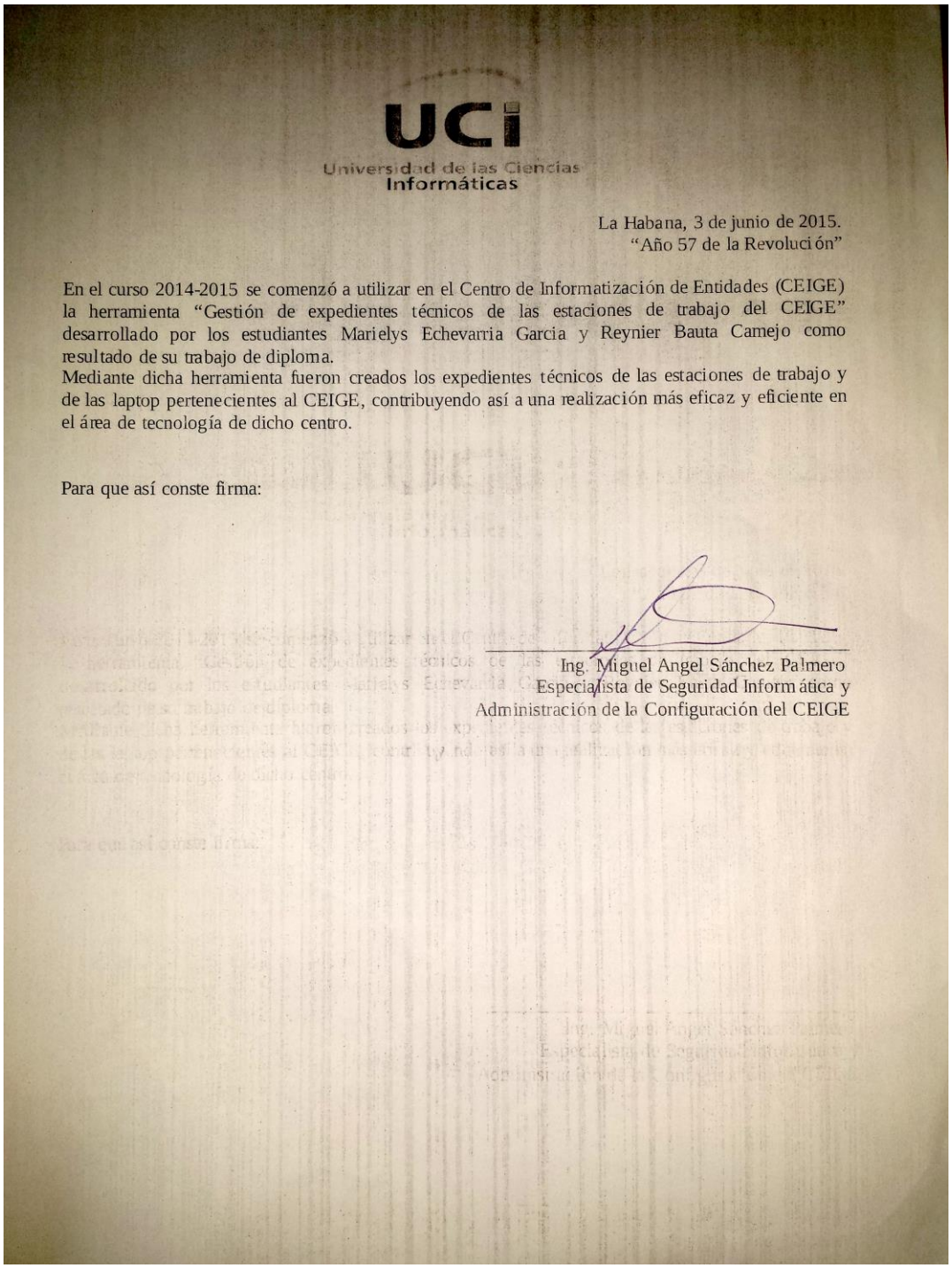
Para que así conste firma:

Ing Yoandy Perez Villazón
Director de CESOL

Ing. Alfredo Pérez Benitez
Metodólogo en Tecnología y Seguridad Informática

UCI Universidad de las Ciencias Informáticas
Centro CESOL Facultad 1

Anexo 7: Aval emitido por el Especialista de Seguridad Informática y Administración de la Configuración del CEIGE



UCI
Universidad de las Ciencias Informáticas

La Habana, 3 de junio de 2015.
"Año 57 de la Revolución"

En el curso 2014-2015 se comenzó a utilizar en el Centro de Informatización de Entidades (CEIGE) la herramienta "Gestión de expedientes técnicos de las estaciones de trabajo del CEIGE" desarrollado por los estudiantes Marielys Echevarria Garcia y Reynier Bauta Camejo como resultado de su trabajo de diploma.
Mediante dicha herramienta fueron creados los expedientes técnicos de las estaciones de trabajo y de las laptop pertenecientes al CEIGE, contribuyendo así a una realización más eficaz y eficiente en el área de tecnología de dicho centro.

Para que así conste firma:

Ing. Miguel Angel Sánchez Palmero
Especialista de Seguridad Informática y
Administración de la Configuración del CEIGE

Anexo 8: Cuestionario aplicado para validar la investigación.

Cuestionario aplicado al personal que trabajan con la Herramienta de la gestión de expedientes técnicos del centro CEIGE (GETET).

En este cuestionario se evalúa el aporte del desarrollo de la herramienta de la gestión de expedientes técnicos del CEIGE en el aumento del control del *hardware* sobre las estaciones de trabajo por esta razón es importante que su respuesta sea con honestidad. Evalúe según el nivel de aceptación de menor a mayor, en un rango de valores del 1 (menor) al 5 (mayor) lo que piensa en las siguientes preguntas.

Usted fue seleccionado para participar en esta encuesta pues por su cargo o responsabilidad en el centro en algún momento tuvo que interactuar con los inventarios de *hardware* realizados.

Antes de la aplicación de la herramienta GETET.

1. ¿Es óptima la obtención del levantamiento de la información del *hardware* de las estaciones de trabajo?
1___ 2___ 3___ 4___ 5___

2. ¿La forma de registrar la información del hardware de las estaciones de trabajo es la más factible?
1___ 2___ 3___ 4___ 5___

3. ¿La información de las estaciones de trabajo está correctamente compartimentada?
1___ 2___ 3___ 4___ 5___

Nota: *Compartimentada: Término utilizado para referirse una distribución personalizada de la información entre las áreas.*

4. ¿La elaboración de las fichas técnicas y los expedientes técnicos de las estaciones de trabajo es la más eficiente?
1___ 2___ 3___ 4___ 5___

5. ¿Es posible realizar análisis de la información registrada con el mínimo de esfuerzos?
1___ 2___ 3___ 4___ 5___

Después de la utilización de la herramienta GETET.

1. ¿Es óptima la obtención del levantamiento de la información del *hardware* de las estaciones de trabajo?
1___ 2___ 3___ 4___ 5___

2. ¿La forma de registrar la información del *hardware* de las estaciones de trabajo es la más factible?
1___ 2___ 3___ 4___ 5___

3. ¿La información de las estaciones de trabajo está correctamente compartimentada?
1___ 2___ 3___ 4___ 5___

4. ¿La elaboración de las fichas técnicas y los expedientes técnicos de las estaciones de trabajo era la más eficiente?
1___ 2___ 3___ 4___ 5___

5. ¿Es posible realizar análisis de la información registrada con el mínimo de esfuerzos?
1___ 2___ 3___ 4___ 5___