Universidad de las Ciencias Informáticas

Facultad 1

Aplicación de escritorio para la selección de alternativas libres al software privativo en las estaciones de trabajo

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

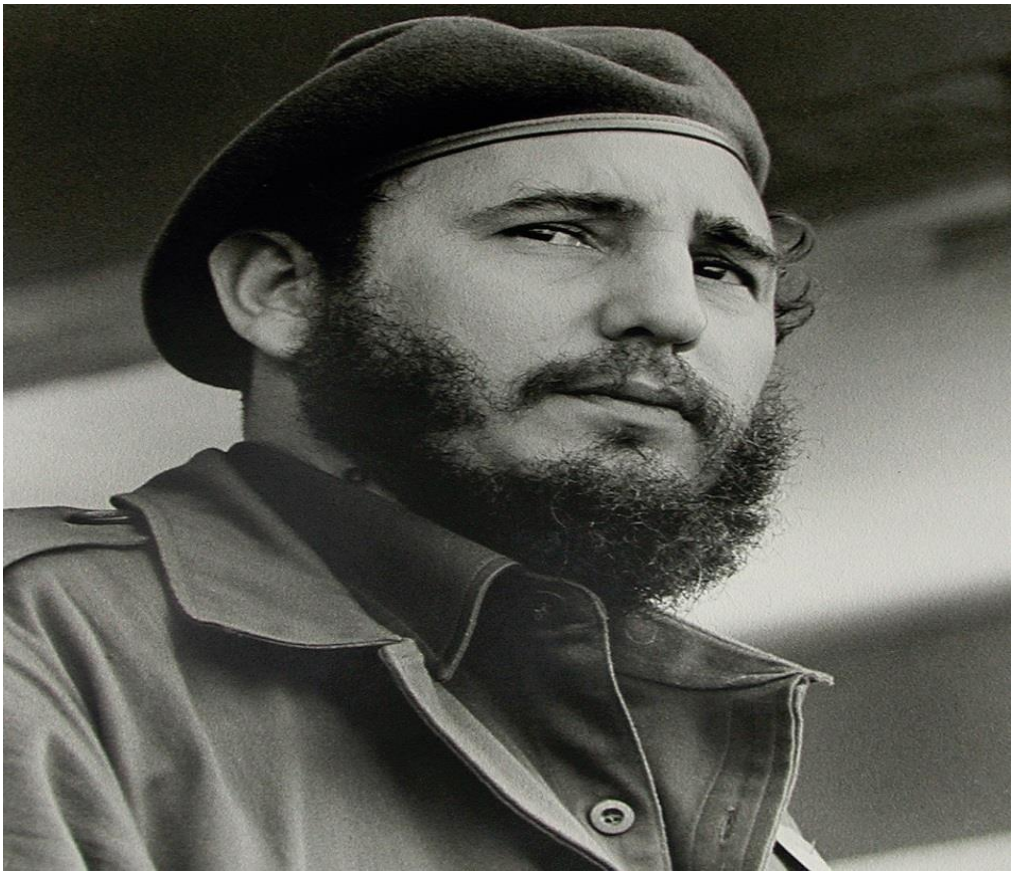
Autor: Nelsón Yaidel Deus Soler

Tutores: Ing. María Leisy González Carrera

Ing. Yasiel Pérez Villazón

La Habana, junio 2017

"Año 59 de la Revolución"



"Sé que la cárcel será dura como no la ha sido nunca para nadie, preñada de amenazas, de ruin y cobarde ensañamiento, pero no la temo, como no temo la furia del tirano miserable que arrancó la vida a 70 hermanos míos. Condenadme, no importa, la historia me absolverá".

Comandante Fidel Castro Ruz

Agradecimientos

A mis tutores por su ayuda incondicional en este período, por haber sido mis mejores guías.

A mis amigos y compañeros de aula por aguantarme por todo este tiempo.

A mis primos que los quiero como hermanos, por confiar en mí y apoyarme en cada reto de la carrera.

A Liliana y Agnes por ser verdaderas amigas y estar siempre conmigo en momentos malos y buenos.

A yanet por apoyarme mucho durante la tesis.

A Javier y Geidar que me ayudaron mucho en la tesis.

Dedicatoria

A mi familia por darme todo su apoyo y comprensión en los momentos más difíciles durante mis estudios.

A mi abuela que aunque ya no se encuentra hoy con nosotros siempre tendrá un lugar muy especial en mi corazón.

Declaración de autoría

Declaro por este medio que yo Nelsón Yaidel Deus Soler, con carné de identidad 92051021863 soy el autor principal del trabajo titulado “Aplicación de escritorio para la selección de alternativas libres al software privativo en las estaciones de trabajo” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de junio del año 2017.

Nelsón Yaidel Deus Soler

_____

Firma del Autor

Ing. María Leisy González Carrera

_____

Firma del Tutor

Ing. Yasiel Pérez Villazón

_____

Firma del Tutor

Resumen

La presente investigación tiene como objetivo principal desarrollar una aplicación de escritorio para *Windows* que permita la selección de alternativas libres al software privativo, durante el proceso de migración a código abierto en las estaciones de trabajo de los Órganos y Organismos de la Administración Central del Estado. Para el estudio del arte se emplearon los métodos científicos analítico-sintético con el propósito de construir los elementos teóricos de la investigación. En el desarrollo de la solución se utilizaron las siguientes tecnologías y herramientas: *Visual Studio* como entorno de desarrollo integrado, C# como lenguaje de programación y la herramienta de modelado *Visual Paradigm*. La metodología de desarrollo de software utilizada es *AUP* en la variante UCI. Para validar el correcto funcionamiento del sistema se realizaron dos tipos de prueba, unitarias y funcionales arrojando como resultado en tres iteraciones un total de 15 no conformidades a las que se le dieron solución. Como resultado final se obtuvo una aplicación de escritorio que permite al especialista en migración realizar un inventario de las aplicaciones privativas instaladas en las estaciones de trabajo de la institución durante el proceso de migración a código abierto. Además propone las alternativas libres más adecuadas al software privativo, teniendo en cuenta atributos de calidad. La aplicación debe ser ejecutada en el sistema operativo *Windows*, a través del uso de una memoria o disco externo. Para ello, se utiliza una base de datos en la que se encuentran almacenadas las alternativas libres y los softwares privativos.

Palabras clave: alternativas libres, atributos de calidad, código abierto, proceso de migración

Índice

Índice de figuras

Índice de tablas

Introducción

A nivel mundial se perciben dos grandes grupos de desarrolladores de software: de software libre y de software propietario. El software libre es una cuestión de libertad, no de precio, se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software (Stallman, 2004). En este sentido el software libre constituye una alternativa a las soluciones privadas o públicas, este se encuentra regido por distintas licencias que facilitan su uso de forma gratuita.

El uso del software libre en la sociedad no solo viene dado por lo que significa en lo económico sino también por el hecho de que brinda una mayor seguridad y fiabilidad. Para llevar a cabo un proceso de soberanía tecnológica es necesario realizar la migración a tecnologías de software libre y código abierto que se define como: un proceso ordenado en el cual se sustituye parcial o totalmente el software existente en la organización por alternativas liberadas bajo licencias libres o de código abierto. Debe incluir la adopción de estándares abiertos para la documentación (Villazón, y otros, 2014).

En la migración de las estaciones de trabajo se afectan todas las estructuras y personas de la institución provocando desconfianza y resistencia al cambio. Para disminuir los índices de fracaso en este proceso es necesario que el personal técnico encargado se encuentre capacitado, ofreciendo a los trabajadores de la entidad una mejor preparación al enfrentar los cambios. También existen cambios en el plano tecnológico cuyo mayor desafío es garantizar la compatibilidad entre los programas y sus alternativas, por lo que es estrictamente necesario garantizar el funcionamiento del ecosistema tecnológico de la entidad durante el proceso de migración (Villazón, y otros, 2014).

De esta manera se puede probar cada una de las aplicaciones que sustenten los procesos de negocio de la institución en la migración social y tecnológica. Para garantizar que en el proceso de migración las personas tengan una menor resistencia al cambio es necesario que se instalen sobre el software privativo las aplicaciones libres que en un futuro se utilizarán sobre el sistema operativo libre (Villazón, y otros, 2014).

Cuba en su afán de lograr la soberanía tecnológica crea en el año 2005 la primera distribución cubana de *GNU/Linux* Nova. Esta distribución fue desarrollada por estudiantes y profesores de la Universidad de las Ciencias Informáticas (UCI) para apoyar la migración del país a tecnologías de software libre y código abierto. Como parte del proceso de informatización de la sociedad cubana, por convenio del Consejo de

Ministros se anunció el acuerdo 084, se orienta la migración paulatina de los Órganos y Organismos de la Administración Central del Estado (OACE) hacia aplicaciones de código abierto.

Para guiar los procesos de migración a código abierto se elabora el libro titulado “Buenas Prácticas para la Migración a Código Abierto”, la cual describe una estrategia para ejecutar las actividades de la migración a código abierto. En el proceso de migración el especialista debe ser capaz de identificar las aplicaciones más adecuadas para el entorno de la institución así como las diferentes configuraciones que pueden minimizar los efectos de la resistencia al cambio de los usuarios (Villazón, y otros, 2014).

En aras de apoyar el proceso se desarrolla la Plataforma Cubana de Migración a Código Abierto (PCMCA), la cual cuenta con el subsistema Directorio de Software (Pérez Guevara, y otros, 2014). Este subsistema almacena perfiles de software con características y funcionalidades propias, sin embargo se detectan las siguientes deficiencias:

- La plataforma no cuenta con continuidad de desarrollo, por lo que las aplicaciones que se proponen como soluciones alternativas libres al software privativo en las instituciones cubanas no presentan en todos los casos una adecuada correspondencia con las necesidades de la institución.
- Las tecnologías que presenta la PCMCA son obsoletas y muchas no son compatibles con nuevas versiones que se deseen desarrollar.
- Para la utilización de la plataforma es necesario tener conexión de red. En ocasiones esto dificulta el trabajo de los especialistas en el proceso de migración, debido a que no todas las instituciones presentan una infraestructura tecnológica capaz de brindar un buen servicio de red.

Partiendo de la situación anterior se plantea el siguiente **problema de la investigación:** ¿Cómo lograr la selección de alternativas libres al software privativo en las estaciones de trabajo durante el proceso de migración a código abierto?

Se define como **objeto de estudio** el proceso de migración y selección de aplicaciones privativas a aplicaciones libres **campo de acción** el proceso de selección de aplicaciones privativas a aplicaciones libres en las estaciones de trabajo.

Se define como **objetivo general:** Desarrollar una aplicación de escritorio para la selección de alternativas libres al software privativo en las estaciones de trabajo durante el proceso de migración a código abierto. Luego se derivan los siguientes **objetivos específicos**:

1. Sistematizar en tecnologías para la selección de alternativas libres al software privativo.
2. Analizar funcionalidades de la aplicación que permita la selección de alternativas libres al software privativo.
3. Diseñar las funcionalidades de la aplicación que permita la selección de alternativas libres al software privativo.
4. Implementar la solución propuesta.
5. Realizar las pruebas funcionales correspondientes a la aplicación de escritorio.

Como parte del desarrollo de la investigación surgen una serie de preguntas científicas, las cuales ayudan en el proceso de desarrollo de la misma:

1. ¿Cuáles son los referentes que sustentan el desarrollo de una aplicación de escritorio para la selección de las alternativas libres al software privativo en las estaciones de trabajo durante el proceso de migración a código abierto?
2. ¿Cuáles son las características y componentes de las herramientas, tecnologías y metodologías existentes que permiten la construcción de la aplicación de escritorio para la selección de las alternativas libres al software privativo en las estaciones de trabajo?
3. ¿Cuál es la estructura que poseen los elementos que deben tenerse en cuenta para realizar el análisis y diseño de la aplicación de escritorio para la selección de las alternativas libres al software privativo en las estaciones de trabajo?
4. ¿Qué contribución tiene la propuesta de solución para la selección de las alternativas libres al software privativo en las estaciones de trabajo durante el proceso de migración a código abierto?

Los métodos científicos utilizados en el proceso de investigación son el **analítico-sintético** y la **modelación**. El método **analítico-sintético** permitió realizar un estudio con mayor profundidad acerca de las alternativas libres al software privativo en las diferentes bibliografías consultadas sobre el tema en cuestión. El método **modelación** se utilizó para la representación de las características del sistema mediante el uso de diagramas y relaciones entre objetos que intervienen en los procesos implementados por la propuesta de solución.

El presente documento se encuentra estructurado de la siguiente forma: introducción, tres capítulos, conclusiones generales, recomendaciones, referencias bibliográficas, anexos.

Capítulo 1. Fundamentación teórica. Se realiza un análisis del tema a tratar, se abordan los conceptos de migración parcial y total, software privativo y de código abierto. Además, se realiza un estudio de las herramientas que realizan inventario de software y directorios de software. Se define la metodología de desarrollo para guiar el proceso de desarrollo de la aplicación a proponer, además de las herramientas y tecnologías a utilizar. Se realiza un estudio de los atributos de calidad que regirá el proceso de selección de las alternativas libres al software privativo.

Capítulo 2. Análisis y diseño. En el presente capítulo se describe la propuesta de solución. Se realiza la especificación de requisitos funcionales y no funcionales. Se ilustra el diagrama de clases del diseño donde se describen las clases y funcionalidades. Además, se define la arquitectura y los patrones de diseño GRASP.

Capítulo 3. Implementación y prueba. Se describe la estructura de la aplicación a través del diagrama de componentes y el estándar de código usado en la implementación. Se define la estrategia de pruebas y los resultados obtenidos en cada una de estas.

.

Capítulo 1: Fundamentación teórica

En el presente capítulo se definen los principales conceptos asociados al tema investigativo como son migración parcial y total, software de código abierto y privativo. Se realiza un análisis de las herramientas, lenguajes y tecnologías a utilizar para el desarrollo de la aplicación y se define la metodología a utilizar.

1.1 Conceptos generales

Para el desarrollo de la presente investigación se tienen en cuenta los siguientes conceptos que son descritos en el proceso de migración:

Migración parcial: es la combinación de migrar solo una parte de los servidores o de los clientes, de manera que continúa existiendo en la institución computadoras con software privativo. Significa que en el mismo sistema se encuentran clientes basados en software libre y privativo. Esta brinda a los usuarios una mayor confianza en el trabajo con aplicaciones libres y una menor resistencia al cambio (Villazón, y otros, 2014).

Migración total: resulta de la combinación de migrar tanto los servidores como las computadoras de los usuarios. La planificación de este tipo de migración tiene que garantizar que los clientes no queden en ningún momento sin acceso a los servicios proporcionados por los servidores. Para ello, se suele realizar en primer lugar la migración total o parcial de los servidores y, a continuación, la migración de las máquinas clientes (Villazón, y otros, 2014).

Código abierto: *Open Source* o código abierto, es la expresión con la que se conoce al software distribuido y desarrollado libremente. Es un movimiento más pragmático, se enfoca más en los beneficios prácticos como acceso al código fuente que en aspectos éticos o de libertad. Su premisa es que al compartir el código, el programa resultante tiende a ser de calidad superior al software propietario. El propietario no estás obligado a publicar su código fuente (omg.org, 2016).

Software privativo: se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo, o cuyo código fuente no está disponible o el acceso a éste se encuentra restringido. Las licencias del software privativo implican la pérdida absoluta de control de parte del usuario, quien se convierte en un dependiente del fabricante del software (GNU, 2016).

1.2 Migración a código abierto de las estaciones de trabajo

En el proceso de migración el especialista debe ser capaz de identificar las aplicaciones más adecuadas para el entorno de la institución así como las diferentes configuraciones que pueden minimizar los efectos de la resistencia al cambio de los usuarios. Una de sus actividades fundamentales es la atención individualizada a los usuarios durante la migración de las estaciones de trabajo, durante la cual debe documentar todas las incidencias de soporte detectadas (Villazón, y otros, 2014).

El diagnóstico a las estaciones de trabajo es la obtención de información útil para la migración sobre las mismas; esta persigue dos objetivos (Villazón, y otros, 2014):

- Obtener información sobre las aplicaciones informáticas instaladas y utilizadas en las computadoras de los usuarios
- Obtener la información de los componentes de hardware de cada estación de trabajo

1.2.1 Atributos de calidad para la selección de las alternativas libres más adecuadas al software privativo

En el libro *Buenas prácticas para la migración a código abierto* se ilustran los procedimientos a seguir para migrar una institución a software de fuentes abiertas. Una etapa importante del proceso es proponer las alternativas libres al software privativo, para ello debe elegirse adecuadamente la solución que mejor se ajuste al entorno. Existen en la actualidad varios modelos descritos para la evaluación y selección del software de código abierto, sin embargo, el país tiene condiciones tecnológicas propias. Por ejemplo, en las instituciones no siempre existe una correcta infraestructura de los servicios de red, las computadoras en muchos casos son muy antiguas con un rendimiento por debajo de las necesidades de la propia institución. Por lo que, los atributos de calidad definidos tienen que estar en constante análisis para sus posibles cambios en el proceso de selección de alternativas libres.

Para la evaluación y selección del software existen atributos particulares en dependencia de la solución privativa de que se está evaluando y de los procesos que la misma automatiza en la institución objeto del proceso de migración.

Para la selección de las aplicaciones de código abierto se propone tener en cuenta los siguientes atributos. Estos atributos mencionados son generales y deben ser aplicados independientemente de la aplicación a la que se seleccione una alternativa (González Carrera, y otros, 2016)

- **Licencia de uso / Distribución**: permite conocer si el producto es realmente de software libre o no, para verificarse se recomienda emplear el sitio de la *Iniciativa para el Código Abierto*, donde se enuncian las licencias que son compatibles con el software libre.
- **Propietario**: permite conocer si el producto es desarrollado por una persona o una institución. Los productos desarrollados por instituciones son más recomendados que los desarrollados por individuos independientes.
- **Procedencia**: no se recomienda productos provenientes de empresas/individuos norteamericanos. En caso de ser seleccionados deben ser sometidos a un riguroso proceso de revisión del código fuente.
- **Antigüedad:** los productos más antiguos poseen mayor madurez, por lo que son menos propensos a vulnerabilidades.
- **Cantidad de errores**: el estudio de los errores y su tendencia permite conocer los softwares que han incrementado su confiabilidad en el tiempo.
- **Soporte al idioma español**: es prioritario que la solución a seleccionar esté disponible en el idioma Español.
- **Impacto, grado de uso**: las aplicaciones más usadas poseen por lo general una mayor aceptación de los usuarios.
- **Comunidad / Rapidez en el desarrollo**: la solución a seleccionar debe poseer un desarrollo activo, si alrededor de una aplicación existe una comunidad debe ser un elemento de peso a tener en cuenta. Las comunidades activas garantizan la mantenibilidad en el tiempo del producto y mejores tiempos de respuesta ante posibles problemas.
- **Rendimiento:** por las condiciones tecnológicas de Cuba se recomiendan aquellas aplicaciones que realicen un mejor aprovechamiento de hardware. Esta condición es prevaleciente mientras no afecte la productividad final.
- **Documentación**: el grado de disponibilidad de la documentación debe ser tenido en cuenta, aquellas aplicaciones poco documentadas, por lo general son complejas y poco usadas. La existencia de foros de discusión, listas de correo, manuales de uso y de desarrollo son elementos a tener en cuenta.

- **Disponibilidad de las pruebas**: las aplicaciones libres cuyas pruebas aplicadas estén disponibles y puedan ser ejecutadas.
- **Soporte:** existen varios niveles de soporte, los más populares para productos de fuentes abiertas son el soporte comunitario y el profesional. Se debe tener en cuenta que exista un adecuado soporte comunitario. Este atributo está vinculado a la existencia de una comunidad.

1.3 Análisis de las aplicaciones de levantamiento de información

Para el desarrollo del proceso de migración es necesario una aplicación que permita realizar un inventario de software, para ello se debe realizar un estudio de herramientas y aplicaciones con este fin.

OSC Inventory

Herramienta de gestión de inventarios de libre distribución que permite a los usuarios administrar la información recopilada acerca del software y hardware de los equipos en una red. Esta puede utilizarse para visualizar y realizar consultas del inventario a través de una interfaz web y es capaz de detectar dispositivos de la red tales como: *switch*, *routers* e impresoras (ocsinventory, 2016). A continuación se listan algunas características de la herramienta:

- Cuenta con una búsqueda que permite filtrar datos como programas instalados, memoria RAM, redes.
- El agente debe estar instalado y configurado en cada servidor o computadora a la cual se le realizará el inventario.
- Permite agrupar los servidores por diferentes criterios (similar a la búsqueda).
- La herramienta se encuentra disponible para *Windows* y *Linux*.
- Permite el análisis de la red.

GLPI[1]

Herramienta de software libre distribuida bajo licencia GPL[2], facilita la administración de recursos informáticos. Permite registrar y administrar los inventarios del hardware y software de una empresa, optimizando el trabajo de los técnicos. Incluye software de ayuda para el registro y atención de solicitudes

[1] GPLI: es una solución libre de gestión de servicios de tecnología de la información.

[2] GPL: Es la licencia de derecho de autor más ampliamente usada en el mundo del software libre y código abierto.

del servicio de soporte técnico, con posibilidades de notificación por correo electrónico a los usuarios y al mismo personal de soporte, al inicio, avances o cierre de una solicitud (glpi-project.org).

Las principales funcionalidades están articuladas sobre dos ejes: a) el inventario preciso de todos los recursos informáticos y el software existente, cuyas características se almacenan en bases de datos y b) la administración e historiales de las diferentes labores de mantenimiento y procedimientos relacionados, llevados a cabo sobre los recursos informáticos.

Open-AudIT

La aplicación muestra lo que existe en la red, cómo se configura y cuándo cambia. Se ejecuta en sistemas *Windows* y *Linux*. Permite almacenar información en una base de datos, que se puede consultar a través de una interfaz web. Los datos sobre la red se insertan a través de un *bash*[3] *script (Linux)* o *vbscript*[4] *(Windows).* La aplicación está escrita en *php, bash, y vbscript*. Permite realizar cambios y personalizaciones, es rápido y fácil. Los dispositivos de red (impresoras, conmutadores, *routers*) pueden tener datos grabados como dirección IP, dirección MAC, puertos abiertos y número de serie. La salida está disponible en *pdf, csv*[5] y páginas web. Existen opciones de exportación para Dia[6] e Inkscape[7] (open-audit.org, 2017). Se puede configurar para escanear la red y dispositivos automáticamente y se recomienda una exploración diaria para los sistemas, por ejemplo exploraciones de red cada dos horas. De esta forma, se notifica si existe algún cambio en la computadora o si aparece algo nuevo en la red.

Fusion Inventory

Tiene la función de obtener la información del software y *hardware* instalado en sus computadoras. También dispone de un *plugin* que funciona con GLPI y sirve para transferir la información obtenida de cada máquina al GLPI (FusionInventory, 2016). Para ello cuenta con:

[3] Bash: es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de programación de consola.

[4] *Vbscript:* son lenguajes de script son versiones recortadas de otros lenguajes.

[5] *csv*: son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla.

[6] Dia: es una aplicación informática de propósito general para la creación de diagramas

[7] Inkscape: es un editor de gráficos vectoriales gratuito y de código libre

- Un agente *(Fusion Inventory Agent)* que se instala en cada computadora. Funciona tanto en entornos *Linux* como *Windows* y está programado en *Perl*[8], además es muy simple la instalación y configuración.
- Un *plugin* que se agrega a GLPI, luego recibe la información de los agentes instalados en las computadoras y tiene la capacidad de escanear las redes a las que se encuentra conectado descubriendo impresoras y otros hardwares / softwares conectados a sus redes.

1.4 Análisis de los directorios de software

Para la correcta selección de una alternativa libre es necesario tener en cuenta las características principales de cada software y la forma en que se encuentran estructurados, para ello es necesario realizar un estudio de los directorios de software que nos muestran esos detalles.

Free Software Directory

El Directorio de Software Libre es un proyecto de la *Free Software Foundation* (FSF por sus siglas en ingles) y la Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura (UNESCO). Posee una serie de catálogos de software libre que son útiles, ya que se ejecuta en sistemas operativos libres GNU/Linux y sus variantes. Posee más de 5 000 programas de software libre (directory.fsf.org, 2016). Posee varias características entre las que se encuentran:

- La información está organizada por categorías y subcategorías.
- Posee un buscador de aplicaciones libres
- No brinda alternativas a programas privativos.
- Ofrece información sobre cada aplicación como: la página oficial, licencia, la valoración de los usuarios.
- Permite evaluar las aplicaciones

Open Source as alternative

Está disponible en idioma español. Es una aplicación web bastante intuitiva y muy básica, se centra en proveer de una lista de software privativo y libre, para así poder hacer una comparación y valoración entre

[8] Perl: es un lenguaje de programación

ellos. La idea es encontrar el software libre que compite con determinado software privativo, así el usuario tiene la posibilidad de observar las cualidades de cada aplicación (osalt.com, 2016). Algunas de sus características son:

- Brinda aplicaciones equivalentes a programas para Windows, Mac y Linux.
- Permite buscar las aplicaciones libres, así como las alternativas a los programas privativos mediante un buscador.
- Está organizado por categorías.
- Sugiere las aplicaciones más populares en cada categoría.
- Posibilita la descarga de programas.
- Muestra información sobre cada aplicación como: sistemas operativos compatibles y página web oficial.
- Se puede sugerir o solicitar las aplicaciones sean de código abierto o privativo.

Alternativas Libres

La herramienta establece relaciones entre aplicaciones privativas y las alternativas libres que existen. Ha sido desarrollado exclusivamente usando software libre. La mayor parte del sitio está escrito en PHP, utilizando como servidor web Apache. Todo el código está escrito con Vim[9]. El gestor de bases de datos usado es *PostgreSQL*. Para algunas tareas puntuales se ha utilizado *Perl* y la biblioteca LWP (freealts.com). Posee varias características entre las que se encuentran:

- Tiene un buscador de aplicaciones libres y otro para la obtención de las alternativas libres a los programas privativos.
- Está organizado por categorías.
- El sitio posee dos modos: edición y usuario. En modo edición se puede contribuir al sitio usando los enlaces de edición de cada página, mientras que el modo usuario es utilizado para consultar la información disponible.
- Brinda información sobre cada aplicación como: descripción, página web oficial, valoración de la aplicación y la imagen de la aplicación.

[9] Vim: es un editor de texto

- Permite contribuir al sitio añadiendo aplicaciones libres, aplicaciones privativas o mediante el uso completo de aplicaciones ya disponibles.

1.4.1 Valoración de las herramientas homólogas

A continuación se realiza una comparación de las herramientas anteriormente descritas teniendo en cuenta criterios que cumplan con las funcionalidades para el sistema que se pretende desarrollar:

Servicios de red: las aplicaciones realizan sus servicios utilizando la conexión por red.

Categorías y subcategorías: la información de los softwares se encuentran estructurados en categorías y subcategorías.

Software o hardware: se muestra el listado de las características de hardware y software de las aplicaciones instaladas en las computadoras.

Actualización: la herramienta se encuentra actualizada y las distintas versiones están disponibles a los usuarios.

Tabla 1. Comparación de las herramientas

Criterios	Ocs Inventory	GLPI	Open-AudIT	Fusion Inventory	Free Software Directory	Open Source as alternative	Alternativas Libres
Servicios de red	si	si	si	si	si	si	si
Categorías y subcategorías	no	no	no	no	si	si	si
Software y hardware	no	no	no	no	si	si	si
Actualización	si	si	si	si	si	si	si

Luego de estudiar los sistemas de gestión de inventarios y analizar los directorios de software referente a la bibliografía consultada se puede analizar que los sistemas estudiados realizan sus funciones utilizando los servicios de red lo cual constituye un obstáculo en las instituciones donde no se brinde servicio de red, o la que se brinde no soporte estas aplicaciones. Los directorios de software muestran características bien detalladas de los softwares libres y privativos. Los sistemas de gestión de inventarios nos muestra el listado de las características de hardware y software de las aplicaciones instaladas en las computadoras.

Ambas presentan como mayor dificultad que dada las características de una computadora no es capaz de brindar una alternativa libre al software privativo, por lo que no se escoge ninguna de ellas como propuesta de solución. Por lo anteriormente expuesto se decide realizar una nueva aplicación de escritorio que permita la selección de las alternativas libres al software privativo en la migración de las estaciones de trabajo.

1.5 Metodología de desarrollo

La calidad del producto final está determinada en gran medida por el uso correcto de la metodología de desarrollo. Una metodología de desarrollo de software es un conjunto de pasos, técnicas y procedimientos que deben seguirse para desarrollar software. Las metodologías de desarrollo de software facilitan la planificación, control, seguimiento y evaluación de los proyectos, optimizan la gestión de los recursos y garantizan en la medida de lo posible la calidad del producto (Laboratorio Nacional de Calidad, 2009).

Existen dos clasificaciones fundamentales de las metodologías de desarrollo de software: las metodologías ágiles y las tradicionales. Las metodologías ágiles proponen mejorar la calidad del producto de software a través de la comunicación inmediata y directa con el cliente, mientras que las metodologías tradicionales proponen que sea a través del orden y la documentación.

La aplicación que se propone es de poca documentación, con un tiempo estimado medio y de constante interacción con el cliente, por lo que se decide utilizar una metodología ágil.

1.5.1 Metodologías ágiles

Aplicables a proyectos que necesitan solución rápida y no generan mucha documentación, pero resuelven los problemas de diseño y construcción de manera ágil respecto a las metodologías tradicionales. Su filosofía se centra en el individuo, la colaboración con el cliente y que este sea partícipe del proceso de desarrollo del software, y en el desarrollo incremental del producto software en sí con iteraciones cortas.

1.5.2 Metodología Aup variante UCI

Para lograr la mayor eficiencia en el ciclo de vida de la actividad productiva en la UCI se decidió una variación en la metodología AUP[10] (Rodríguez Sanchez, 2015). La metodología consta de 3 fases:

Inicio: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información

[10] AUP: proceso unificado ágil, describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software

fundamental acerca del alcance de este, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no.

Ejecución: en esta fase se ejecutan las actividades requeridas para desarrollar el software incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se realizan pruebas al producto.

Cierre: en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Para el desarrollo de la propuesta de solución se utiliza el escenario 4 de historia de usuario, debido a que el proyecto fue evaluado y como resultado se obtuvo un negocio bien definido. El cliente está siempre acompañado al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. La duración del proyecto no es muy extensa, y no se debe generar mucha información.

1.6 Tecnologías y herramientas de desarrollo

Para el desarrollo de la aplicación es necesaria la utilización de algunas herramientas que se estudian a continuación.

1.5.1 Lenguaje C#

Lenguaje de programación orientado a objetos, desarrollado y estandarizado por *Microsoft* como parte de su plataforma .NET[11] es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común. Su sintaxis básica se deriva de C/C++ y utiliza el modelo de objetos de la plataforma, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes. Aunque C# forma parte de la plataforma .NET, esta es una API[12] mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma (Microsoft, 2015). Para el desarrollo de la aplicación a desarrollar se utilizará C# en la versión 6.0.

[11] .NET: es un *framework* de *microsoft* que hace un énfasis en la transparencia de redes

[12] API: es un conjunto de subrutinas, funciones y procedimientos en la programación orientada a objetos.

1.5.2 Microsoft Visual Studio 2015

Entorno de desarrollo integrado (IDE, por sus siglas en inglés) para los sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como: C++, C# y Visual Basic. Permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y consolas (Microsoft, 2016).

1.5.3 Visual Paradigm 8.0

Herramienta de UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML. Presenta licencia gratuita y comercial, y es fácil de instalar y actualizar.

Se decide utilizar esta herramienta CASE como medio para modelar el proyecto ya que tiene grandes ventajas entre las que se encuentran (software.com, 2016):

- Soporte de UML.
- Modelado colaborativo con CVS y *Subversion* (control de versiones).
- Interoperabilidad con modelos UML2.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Diagramas de flujo de datos.
- Generador de informes.
- Licencia gratuita y comercial.
- Compatible entre ediciones.

1.5.4 Lenguaje de Modelado

El lenguaje unificado de modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el *Object Management Group* (OMG). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema y aspectos concretos como

expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. Es el lenguaje en el que está descrito el modelo (omg.org, 2016).

1.5.5 Base de datos SQLite

Sistema de gestión de bases de datos relacional compatible con ACID[13], escrita en C. Es un proyecto de dominio público creado por D. Richar Hipp. A diferencia de lo sistema de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, su biblioteca se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos) son guardados como un sólo fichero estándar en la máquina *host*. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción (Creately.com).

1.5.6 ForeUI 4.2

Herramienta de creación de prototipos de interfaz de usuario diseñada para crear prototipos para cualquier aplicación o sitio web. Permite diseñar el comportamiento de prototipos definiendo diagramas de flujo intuitivos para manejar eventos específicos. Su prototipo puede ser exportado a imágenes *wireframe*[14], documentos pdf o simulación HTML5. Es posible utilizar *ForeUI* como IDE para el desarrollo de aplicaciones web reales y funciona en sistemas *Windows*, *Mac*, *OS X*, *GNU/Linux* (foreui.com,2016).

Conclusiones parciales

La descripción de los conceptos generales acerca de la migración parcial y total, software de código abierto y privativo permite una mayor comprensión en el estudio de los procesos de migración. En la comparación de los directorios de software y de las aplicaciones que obtienen inventarios en la red se

[13] se dice que es *ACID* cuando este permite realizar transacciones.

[14] Algoritmo de renderización del que resulta una imagen semitransparente

concluye que es necesaria la implementación de una nueva aplicación que esté acorde a las necesidades de los procesos de migración. La metodología AUP en su variante UCI se ajusta a las condiciones de desarrollo de la aplicación, debido a que la duración del proyecto no es muy extensa y no se debe generar mucha información.

Capítulo 2: Análisis y Diseño

En el presente capítulo se establecen las principales características con que cuenta la solución, ya sean requisitos funcionales y no funcionales, describiendo las historias de usuario correspondientes. Se generan los diagramas de diseño según propone la metodología de desarrollo AUP variación UCI. Además, se describe la arquitectura y los patrones de diseño a aplicar.

2.1 Propuesta de solución

La propuesta de solución es una aplicación de escritorio (ver Figura 1) que permite al especialista en migración mostrar el listado de los softwares privativos que se encuentran instalados en las computadoras de la institución durante el proceso de migración a código abierto y proponer la alternativa libre más adecuada, teniendo en cuenta atributos de calidad. La aplicación debe ser ejecutada en el sistema operativo *Windows*, a través del uso de una memoria o disco externo. Para ello, se utiliza una base de datos en la que se encuentran almacenadas las alternativas libres y los softwares privativos.

La aplicación cuando se ejecuta muestra propiedades de la computadora, tales como: memoria RAM, nombre, gráfico y microprocesador. Permite modificar, adicionar, eliminar, mostrar detalles y mostrar el listado de las alternativas libres que se encuentran en la base de datos, así como la gestión de los softwares privativos. El especialista en migración puede adicionar, modificar y ver el listado de atributos de calidad que se encuentran adicionados en la base de datos con anterioridad. Por defecto se encuentran registrados 12 atributos de calidad que se deben tener en cuenta a la hora de adicionar una alternativa libre.

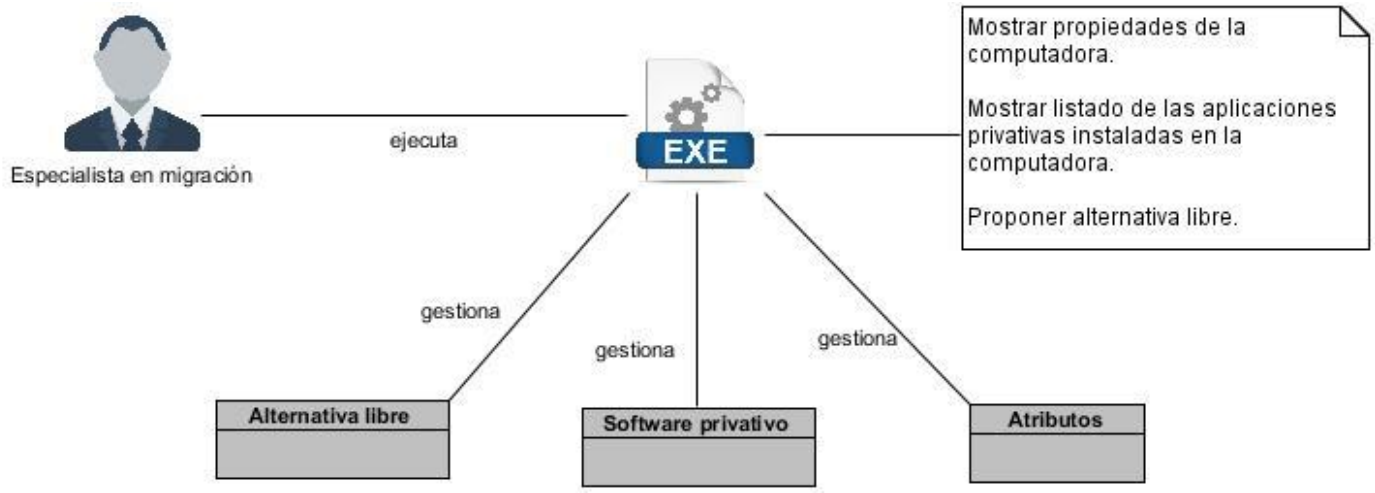


Figura 1: Propuesta de solución

2.2 Requisitos funcionales

La aplicación de escritorio cuenta con total de 16 requisitos funcionales y 3 no funcionales, donde 2 son de prioridad alta, 11 de prioridad media y 3 de prioridad baja. En la siguiente tabla se recogen los requisitos funcionales del producto.

Tabla 2: Requisitos funcionales

No	Nombre	Descripción	Prioridad	Complejidad
RF1	Mostrar listado de los softwares privativos	Debe mostrar el listado de los softwares privativos que se encuentran instalados en la computadora.	Alta	Baja
RF2	Mostrar listado de las alternativas libres	Debe mostrar el listado de las alternativas libres al software privativo que se encuentra instalado en la computadora.	Alta	Baja
Gestión de atributos de calidad				
RF3	Adicionar atributo	Debe adicionar un atributo que permita la selección de la alternativa libre por parte del especialista en migración.	Media	Media
RF4	Editar atributo	Debe editarse un atributo previamente adicionado en la aplicación.	Media	Media

RF5	Mostrar listado de atributos de calidad	Debe mostrar el listado de atributos previamente adicionados en la aplicación.	Media	Media
Gestión de *software* libre				
RF6	Adicionar software libre	Debe adicionar un software libre para el análisis de la mejor alternativa.	Media	Media
RF7	Eliminar software libre	Debe ser capaz de eliminar un software ya existente en la base de datos.	Baja	Baja
RF8	Editar software libre	Debe modificar el software previamente insertado en la base de datos	Media	Media
RF9	Mostrar listado de software libre	Debe mostrar el listado de software libres de la base de datos	Media	Media
Gestión de software privativo				
RF10	Adicionar software privativo	Debe adicionar un software privativo para el posterior análisis de los softwares privativos de la computadora.	Media	Media
RF11	Editar software privativo	Debe modificar el software previamente insertado en la base de datos.	Media	Media
RF12	Eliminar software privativo	Debe ser capaz de eliminar un software ya existente en la base de datos.	Baja	Baja
RF13	Mostrar listado de software privativo	Debe mostrar el listado de software privativos de la base de datos	Media	Baja
Información del sistema				
RF14	Mostrar información del sistema	Debe mostrar la información referente a las propiedades de la computadora donde fue ejecutada la aplicación.	Baja	Media
RF15	Actualizar la información del software privativo.	Debe actualizar la información del software privativo en la base de datos.	Media	Baja
RF16	Actualizar la información de las alternativas libres.	Debe actualizar la información de las alternativas libres en la base de datos.	Media	Media

2.3 Requisitos no funcionales

Los requisitos no funcionales se refieren a cualidades que imponen restricciones en el diseño y la implementación. En el proceso de confección de la aplicación se identificaron los siguientes requisitos no funcionales:

Usabilidad

RNF.1 La aplicación debe permitir acceder a sus diferentes interfaces con una profundidad máxima de 2 clics.
RNF.2 El sistema debe poseer interfaces amigables, con botones que tengan nombres sugerentes para que usuarios inexpertos puedan interactuar fácilmente con el software.

Eficiencia

RNF.3 La aplicación debe responder en un tiempo menor de 3 segundos a las solicitudes de los usuarios.

Hardware

RNF.4 Esta debe ser capaz de ejecutarse desde una memoria o un disco externo con un almacenamiento como mínimo de 500 megas.

Software

RNF.5 La aplicación de escritorio debe ser ejecutada sobre el sistema operativo W*indows* (Vista, Seven, 8,10).

2.4 Historias de usuario

Las historias de usuario son la técnica utilizada para especificar los requisitos del sistema. Las mismas guían el proceso de pruebas al sistema y son utilizadas para estimar el tiempo de desarrollo del producto (Patricio , 2003). A continuación se describen 4 historias de usuario, de las cuales 2 son de prioridad alta, una media y una baja. Las demás historias de usuario se describen en los anexos.

Tabla 3: HU Mostrar listado de los softwares privativos

Historia de usuario	
Número: 1	**Nombre del requisito:** Mostrar listado de los software privativos

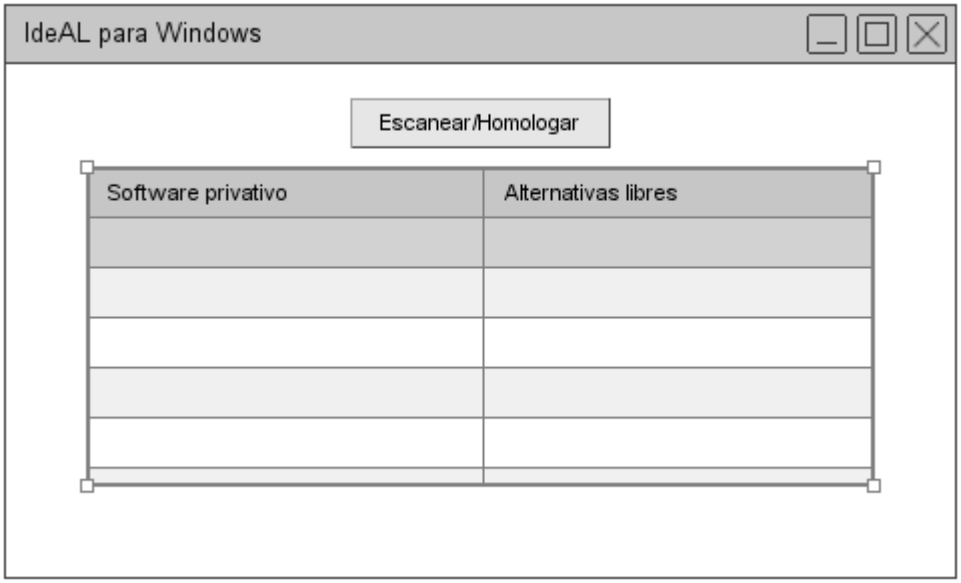
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 120 horas
Riesgo en Desarrollo: No se puede proponer una alternativa libre.	**Tiempo Real:** 120 horas
Descripción: El sistema muestra el listado de los softwares privativos que se encuentran instalados en la computadora de la institución a migrar. El listado muestra el nombre de los softwares instalados.	
Observaciones: Debe ser ejecutada sobre el sistema operativo Windows (Vista, Seven, 8,10).	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 4: HU Mostrar listado de las alternativas libres

Historia de usuario	
Número: 2	**Nombre del requisito:** Mostrar listado de las alternativas libres
Programador: Nelson Yaidel Deus Soler	**Iteración Asignada:** 1

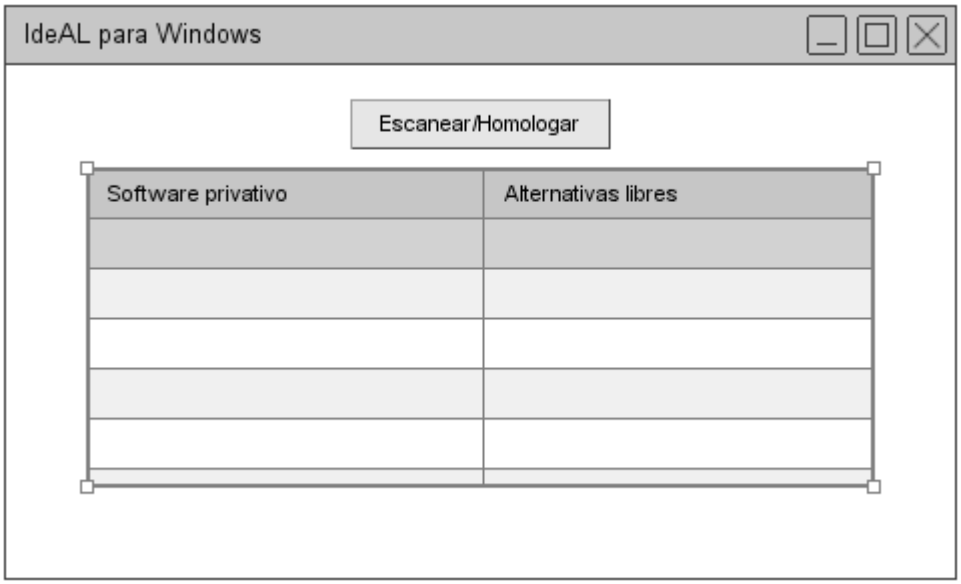
Prioridad: Alta	**Tiempo Estimado:** 120 horas
Riesgo en Desarrollo: No se encuentra la información correspondiente a la alternativa libre.	**Tiempo Real:** 120 horas
Descripción: El sistema muestra el listado de las alternativas libres, una vez que el especialista en migración realice la selección de la alternativa teniendo en cuenta la prioridad de los atributos de calidad. El listado muestra el nombre de la alternativa libre seleccionada.	
Observaciones: Puede que no exista una alternativa libre al software privativo.	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 5: HU Adicionar atributo

Historia de usuario	
Número: 3	**Nombre del requisito:** Adicionar atributo
Programador: Nelson Yaidel Deus Soler	**Iteración Asignada:** 1

Prioridad: Media	**Tiempo Estimado:** 100 horas
Riesgo en Desarrollo	**Tiempo Real:** 100 horas

Descripción: El sistema permite adicionar un atributo de calidad para realizar la selección de las alternativas libres al software privativo. Para ello, se debe tener en cuenta los siguientes atributos:

Nombre: es un campo de texto obligatorio. Permite solo letras comenzando con mayúscula y el carácter espacio en blanco. El máximo de caracteres a introducir es de 50.

Los atributos a tener en cuenta para realizar la selección de la alternativa libre son los siguientes, estos son opcionales:

Licencia de uso/distribución: pueden ser GPL, AGPL, BSD y el tipo campo es un string.

Propietario: el tipo de campo es un string.

Procedencia: el tipo de campo es un string.

Antigüedad: el tipo de campo es un entero.

Cantidad de errores: el tipo de campo es un entero.

Soporte al idioma español: se debe seleccionar si, el tipo de campo es un string.

Impacto, grado de uso: puede ser Alto, Medio o Bajo.

Comunidad / Rapidez en el desarrollo: puede ser Amplia y Activa, No muy activa, Ninguna.

Rendimiento: el tipo de campo es un entero.

Documentación: puede ser un Si o No, el tipo de campo es un string.

Disponibilidad de las pruebas: puede ser un Si o No, el tipo de campo es un string.

Soporte: puede ser Ambos, Comunidad o Profesional.

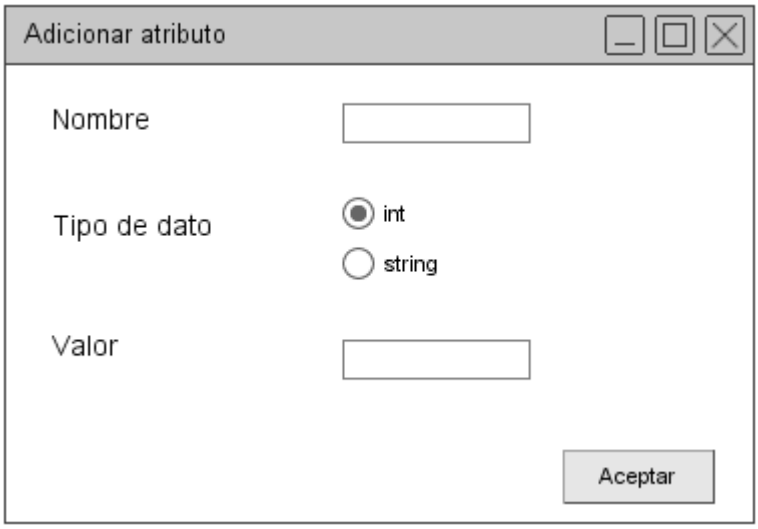
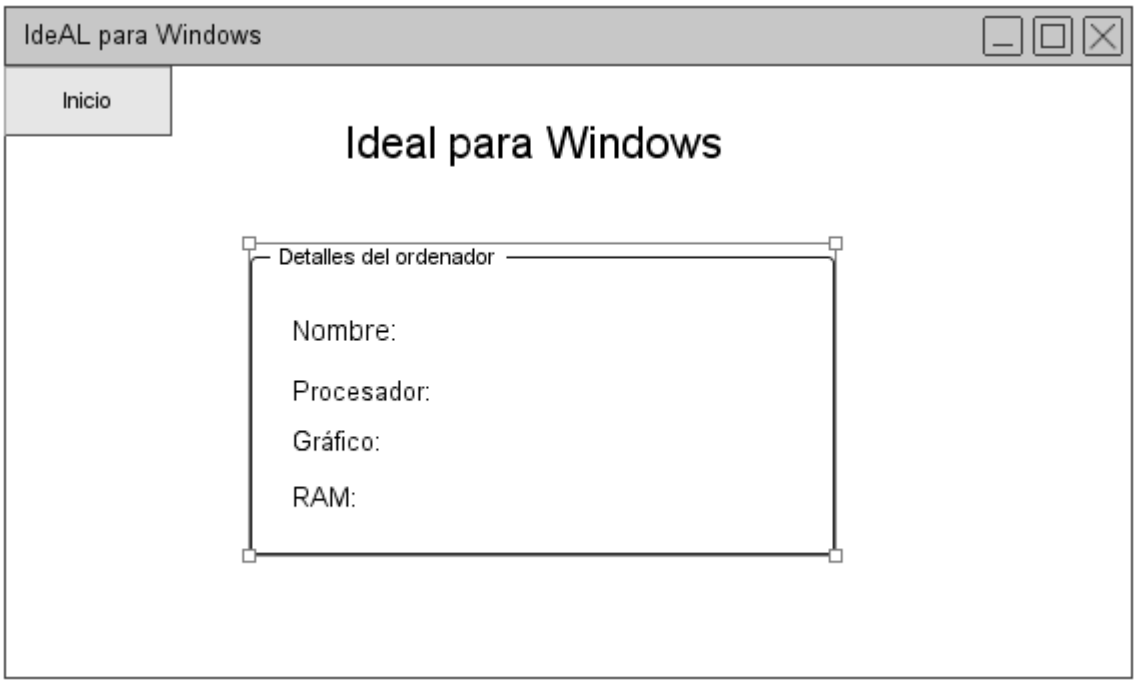
Observaciones: Los especialistas de migración pueden adicionar otros atributos de calidad.
Prototipo elemental de interfaz gráfica de usuario: 

Tabla 6: HU Mostrar información del sistema

Historia de usuario	
Número: 4	**Nombre del requisito:** Mostrar información del sistema
Programador: Nelson Yaidel Deus Soler	**Iteración Asignada:** 1
Prioridad: baja	**Tiempo Estimado:** 90 horas
Riesgo en Desarrollo: ninguno	**Tiempo Real:** 90 horas
Descripción: El sistema debe mostrar la información referente a las propiedades de la computadora donde es ejecutada la aplicación. Las características que se muestran son: tamaño de memoria RAM, gráfico y nombre del microprocesador.	
Observaciones: Estas características influyen directamente en la selección de la alternativa libre, ya que no se puede proponer una aplicación que excede más del rendimiento de la RAM de la computadora.	

Prototipo elemental de interfaz gráfica de usuario



2.5 Arquitectura de diseño

La arquitectura de software proporciona un concepto general del sistema que debe construirse. Describe la estructura y la organización de los componentes del software, sus propiedades y la conexión entre ellos. Además, destaca las decisiones iniciales del diseño y proporciona un mecanismo para considerar los beneficios de estructuras de sistemas alternos (Larman, 2005).

La arquitectura que se propone para la aplicación es la N-capas. Este estilo arquitectónico se basa en una distribución jerárquica de los roles y responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de interacción con otras capas, mientras que las responsabilidades indican la funcionalidad que implementan. Se decide utilizar este estilo arquitectónico ya que permite dividir el proyecto en 3 paquetes, de forma que cada uno de ellos puede trabajar de manera más eficiente. A continuación se muestra la arquitectura a utilizar.

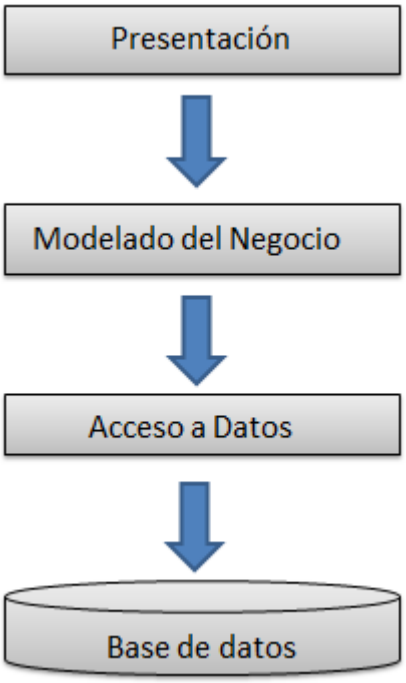


Figura 2: Arquitectura del modelo

Presentación: es la que se encarga de que el sistema interactúe con el usuario y viceversa, muestra el sistema al usuario, le presenta la información y obtiene la información del usuario en un mínimo de proceso.

Modelado del Negocio: es donde residen las funciones que se ejecutan, se reciben las peticiones del usuario, se procesa la información y se envían las respuestas tras el proceso.

Acceso a Datos: es la encargada de almacenar los datos del sistema y de los usuarios. Su función es almacenar y devolver datos.

2.6 Diagrama de paquetes

Para el diseño del diagrama de paquetes se toma como referencia la arquitectura propuesta, donde existe un paquete correspondiente a cada una de las capas. A continuación se muestra una vista detallada de los diagrama de paquetes.

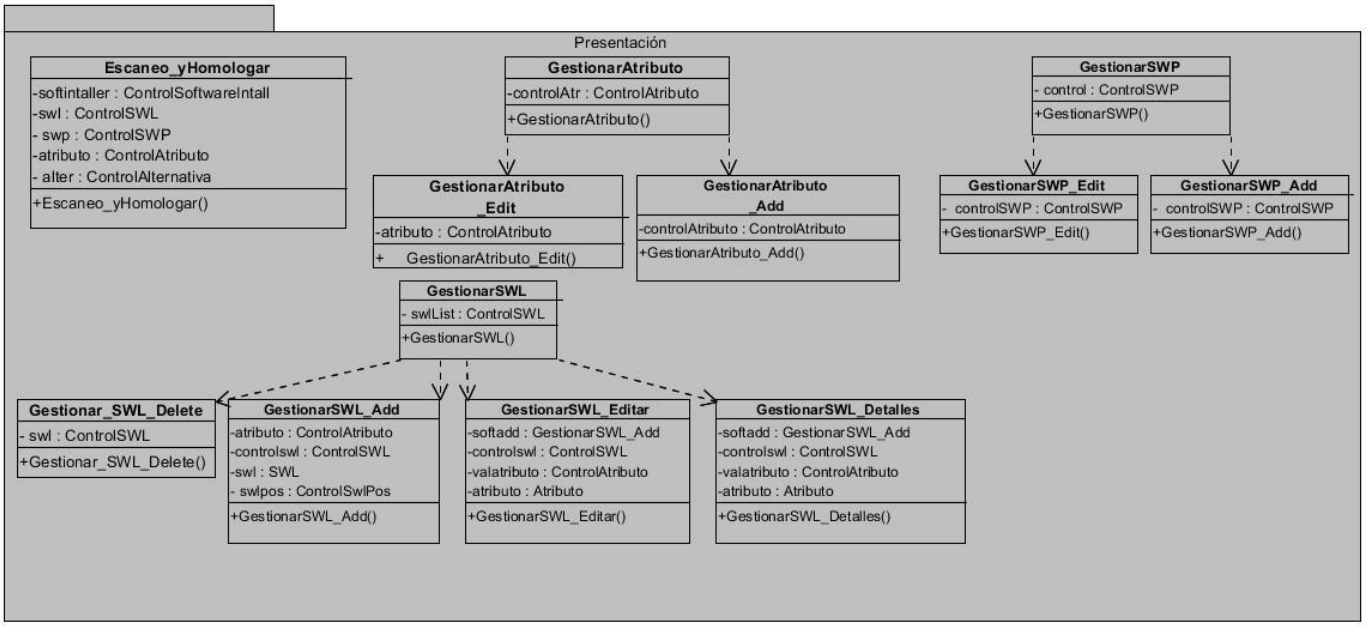


Figura 3: Diagrama de clases de la capa presentación

En la Figura 3 se muestran las clases de la aplicación en la capa de presentación. En esta capa se crean las interfaces con las cuales los especialistas en migración interactúan.

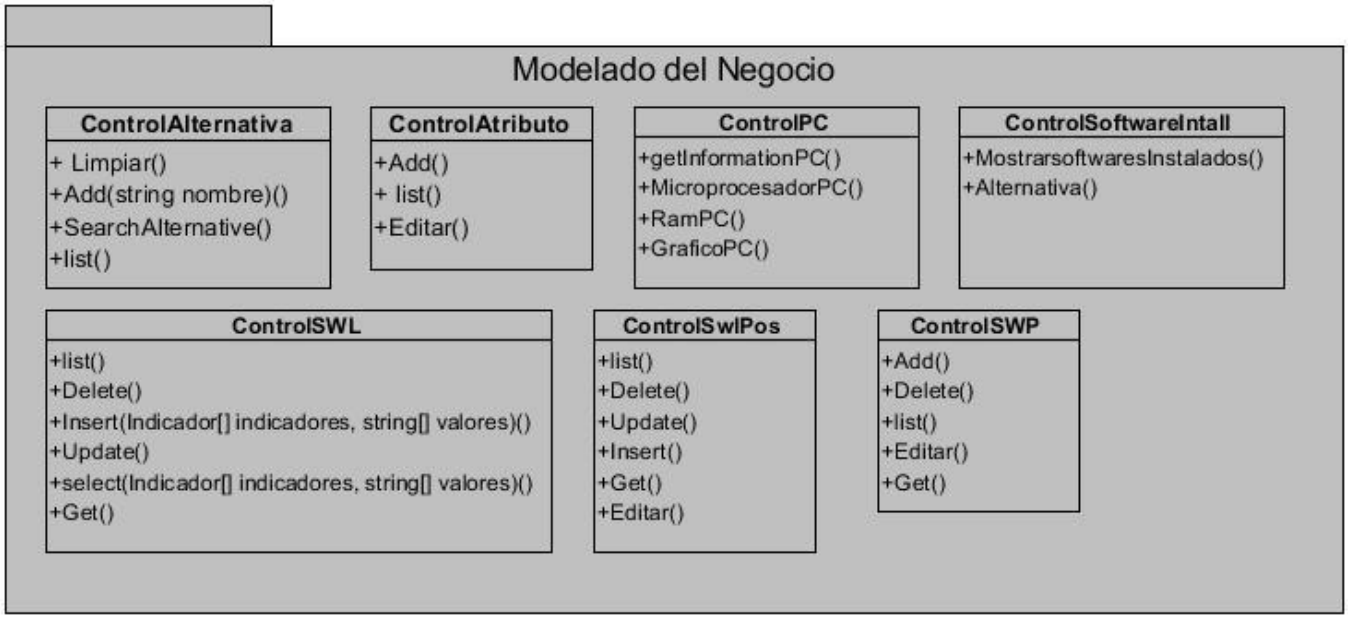


Figura 4: Diagrama de clases de la capa modelado del negocio

En la Figura 4 se muestran las clases de la aplicación en la capa de modelado del negocio. Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados. En esta capa se tiene la clase *ControlAtributo,* que permite el control sobre las distintas funciones de los atributos de calidad, la clase *ControlSWL* gestiona la información de los software libres. En la clase *ControlPC* se gestiona la información de la computadora.

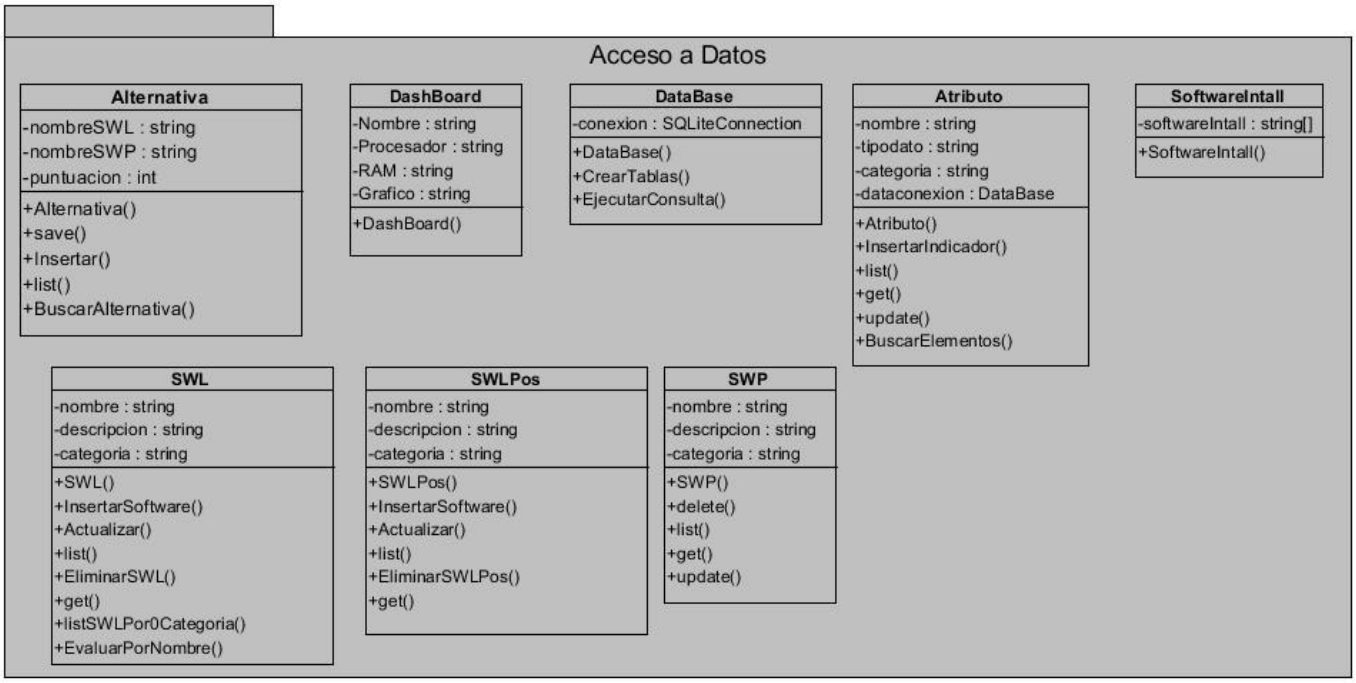


Figura 5: Diagrama de clases de la capa de acceso a datos

En la Figura 5 se muestran las clases de la aplicación en la capa de acceso a datos, estas realizan el almacenamiento de todos los datos. En esta capa se tiene la clase *SWL* que permite conocer los distintos atributos de cada software libre además de las distintas consultas para gestionar los softwares libres. La clase *DataBase* es donde se crean las tablas de las base de datos a utilizar. A continuación se describen algunas de las clases:

Clase ***ControlPC***

Método	Descripción	Parámetros	Valor de retorno
MicroprocesadorPC()	Muestra el nombre del microprocesador.	-	Nombre del microprocesador.
RamPC()	Muestra el tamaño de la RAM.	-	Tamaño de la Ram.

GráficoPC()	Muestra el nombre del gráfico.	-	Nombre del gráfico.

Clase ***ControlSoftwareIntall***

Método	Descripción	Parámetros	Valor de retorno
MostrarsoftwaresInstalados()	Muestra el software instalado en la computadora.	-	Un listado de softwares
Alternativa()	Muestra la alternativa libre a cada software privativo en caso de que la tenga.	-	Listado de alternativas libre a cada software privativo

Clase **SWL**

Método	Descripción	Parámetros	Valor de retorno
List()	Muestra un listado de los software libre de la base de datos.	-	Un listado de software libre.
Delete()	Elimina un software libre.	Nombre : string	-
Update()	Actualiza cada software libre.	Atributos : Atributos[] valores : string [] name : string	-

Insert()	Adiciona un software libre.	Atributos : Atributos[] valores : string []	-

Clase Atributo

Método	Descripción	Parámetros	Valor de retorno
List()	Muestra un listado de los atributos de la base de datos.	-	Un listado de atributos.
Update()	Actualiza cada software libre.	Atributos : Atributos[] valores : string [] name : string	-
InsertarAtributo()	Adiciona un atributo.	Atributos : Atributos[] valores : string []	-

2.7 Patrones de diseño

Los patrones de diseño son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de software. Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. En el modelo de diseño de la aplicación se aplican patrones *GRASP*[15] (Larman, 2005).

[15] GRASP: representan los principios básicos de la asignación de responsabilidades a objetos

2.6.1 Patrones *GRASP*

Estos patrones describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones. A continuación se describen los utilizados en el desarrollo de la aplicación (Larman, 2005).

Experto: se utiliza para la asignación de responsabilidades relacionadas con la obtención de información (Larman, 2005), este patrón se evidencia en la clase *Atributo* que poseen la información y las funcionalidades necesarias para gestionar los atributos.

Creador: guía la asignación de responsabilidades relacionadas con la creación de objetos. La intención básica es encontrar un creador que necesite conectarse al objeto creado en alguna situación (Larman, 2005). Este patrón se pone de manifiesto en la clase *GestionarAtributo* la cual posee instancias de la clase *ControlAtributo.*

Bajo acoplamiento: el acoplamiento es una medida de la fuerza con que un elemento está conectado a otros elementos. Un elemento con bajo o débil acoplamiento no depende de demasiados de otros elementos. Estos elementos pueden ser clases (Larman, 2005). Esto se manifiesta en la relación de las clases *GestionarAtributo- ControlAtributo-Atributo.*

Alta cohesión: es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión. Estos elementos pueden ser clases y subsistemas (Larman, 2005). Esto se manifiesta en la relación de las clases *GestionarAtributo- ControlAtributo-Atributo.*

Conclusiones parciales

La propuesta de solución permitió conocer las principales características y funcionalidades que debe cumplir la aplicación para lograr la selección de alternativas libres al software privativo en las estaciones de trabajo. Se identificaron 16 requisitos funcionales y 5 no funcionales cumpliendo con los requisitos definidos por parte del cliente. La arquitectura utilizada permitió establecer una estructura de las clases de

la aplicación de escritorio, además facilita el manejo de los datos y garantiza que cualquier cambio que se realice en una capa solo la afecte a esta.

Capítulo 3: Implementación y pruebas

En el presente capítulo se describe la estructura de la aplicación a través del diagrama de componentes y el estándar de codificación usado en la implementación de la aplicación. Se establece la estrategia de prueba y se documenta la realización de las pruebas.

3.1 Estándar de codificación

A continuación se muestra el estándar de codificación para la implementación de la aplicación, basado en el estándar del lenguaje C#.

Ficheros de código fuente: mantener las clases y los ficheros cortos, con no más de 2.000 líneas de código y que estén claramente divididas en estructuras.

Ajuste de línea: cuando una expresión no se ajuste en una sola línea de código, dividirla de acuerdo a nueva línea después de una coma.

```
public void  InsertarAtributo(string nombre, string tipodato,
                              string valor)
```

Comentarios de línea: se utilizan para explicar línea a línea el código fuente. También se utilizan para comentar líneas de código temporalmente. Estos comentarios deben tener el mismo nivel de que el código que describen.

```
//consulta a la base de datos swp
string queryInsert = "DELETE FROM swp WHERE nombre='" + this.nombre + "'";
```

Declaraciones de miembros de clases e interfaces: no incluir espacios entre el nombre de un método y los paréntesis donde se encuentran los parámetros del método. La llave de apertura debe aparecer en la línea siguiente a la declaración. La llave de clausura debe comenzar con una línea, alineada verticalmente con su llave de apertura.

```
public DataBase()
{

    if (!File.Exists("software.sqlite")){
        conexion = new SQLiteConnection("Data Source=software.sqlite;Version=3;New=True;Compress=True;");
        this.CrearTablas();
    }
    else {
        conexion = new SQLiteConnection("Data Source=software.sqlite;Version=3;New=True;Compress=True;");
    }

}
```

Sentencias de retorno: una sentencia de retorno no debe utilizar paréntesis para encerrar el valor de retorno.

```
private string MicroprocesadorPC()
{
    ManagementObjectSearcher searcher = new ManagementObjectSearcher("select * from Win32_Processor");

    ManagementObjectCollection queryCollection1 = searcher.Get();
    string microprocesador= "";
    foreach (ManagementObject mo in searcher.Get())
    {

        microprocesador = mo["Name"].ToString();
    }
    return microprocesador;
}
```

Sentencias if, if-else: Debe estar bien estructurado.

```
if (!File.Exists("software.sqlite")){
    conexion = new SQLiteConnection("Data Source=software.sqlite;Version=3;New=True;Compress=True;");
    this.CrearTablas();
}
else {
    conexion = new SQLiteConnection("Data Source=software.sqlite;Version=3;New=True;Compress=True;");
}
```

Convenios de nombres: Estilo PasCal: Este convenio determina que la primera letra de cada palabra debe ser mayúscula.

```
public void EjecutarConsulta(string consulta)
```

Prácticas de programación: Visibilidad: no definir campos públicos; hacerlos privados

3.2 Diagrama de componentes

Un componente representa una parte de un sistema modular, desplegable y reemplazable, que encapsula la implementación y expone un conjunto de interfaces (Ambler, 2010). En el siguiente diagrama se muestran las dependencias que existen entre los componentes de la aplicación.

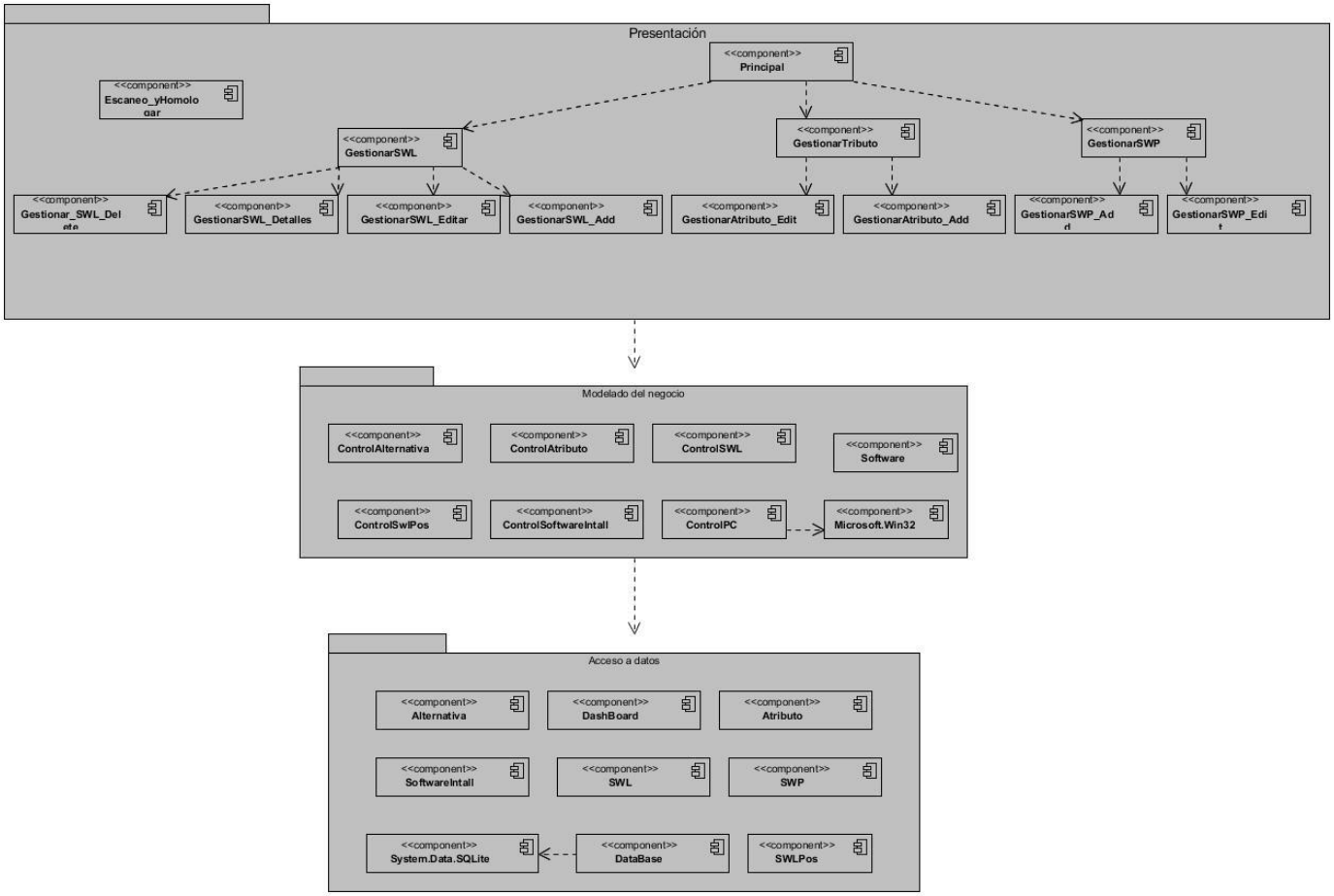


Figura 6: Diagrama de componentes

Los componentes de la aplicación tienen como extensión .cs por las características propias del lenguaje de programación C#. En la capa Modelado del negocio se utiliza la librería *Microsoft.Win32* para realizar las consultas al registro del sistema operativo *Windows*. Además en la capa Acceso a datos se utiliza la librería *System.Data.Sqlite* para el manejo de las consultas en las diferentes clases.

3.3 Mecanismo para proponer las alternativas.

Para la selección de las alternativas se implementó un algoritmo donde se clasifican los valores de los atributos de calidad de las aplicaciones libres. El orden de insertar estos valores es de bueno a malo. A continuación se describen los atributos que se tuvieron en cuenta:

- **Soporte al idioma español**: se le da la posición 0 si presenta soporte al idioma español y 1 lo contrario.
- **Impacto, grado de uso**: se le da la posición 0 si el impacto es alto, 1 si es medio y 2 si es bajo.
- **Comunidad / Rapidez en el desarrollo**: se le da la posición 0 si es amplia y activa, 1 si es no muy activa y 3 si ninguna.
- **Documentación:** se le da la posición 0 si presenta documentación al idioma español y 1 lo contrario.
- **Disponibilidad de las pruebas**: se le da la posición 0 si las pruebas están disponibles y 1 lo contrario.
- **Soporte:** se le da la posición 0 si es ambas (Comunity y Profesional) y 1 si selecciona una de ellas.

En el método *SearchAlternative()* es donde se implementa el algoritmo que da solución al proceso de selección de las alternativas libres al software privativo, a continuación se explica su funcionamiento.

Para definir la mejor alternativa en cada caso es necesario saber la categoría del software privativo de la computadora, para ello se tiene una base de datos de software privativos en la que se pregunta si el software privativo instalado en la computadora se encuentra en la base de datos, en caso de que no este se muestra sin alternativa, si se encuentra pide la categoría de este y se buscan en la base de datos de software libres los softwares que pertenecen a la misma categoría. En caso de no existir ningún software en esa misma categoría se muestra que no tiene alternativa, y en caso de existir un solo software se muestra este como alternativa. Para seleccionar una alternativa en caso de que exista más de un software libre con la misma categoría se procede a buscar por cada software libre el valor de cada atributo en su tabla y el nombre. De la tabla atributo se busca la posición que representa el valor seleccionado, se suma el valor de esa posición a una variable declarada anteriormente, este procedimiento se realiza a cada uno de los software libres que se encuentren en la misma categoría. La variable mientras menor sea, significa que en cada valor del atributo selecciona la mejor opción, por lo que en la tabla alternativa se inserta el

software privativo, el software libre y la menor puntuación. Finalmente se muestra la alternativa libre al software privativo.

3.4 Pruebas de software

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan la excelencia y el desempeño de un software, involucra las operaciones del sistema bajo condiciones controladas y evaluando los resultados. Las técnicas para encontrar problemas en un programa son variadas y van desde el uso del ingenio por parte del personal de prueba hasta herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad (Presman, 2005).

La metodología AUP variante UCI propone la realización pruebas internas, por lo que durante el desarrollo de la aplicación se realiza la siguiente estrategia de pruebas.

3.4.1 Estrategia de prueba

Una estrategia de prueba del software integra los métodos de diseño de casos de prueba en una serie bien planteada de pasos que va a converger en la eficaz construcción del software. La estrategia proporciona un mapa que describe los pasos que se darán como parte de la prueba. Dicha estrategia debe ser lo suficientemente flexible como para promover un enfoque personalizado (Presman, 2005). A continuación se define los tipos de prueba a utilizar en la estrategia de prueba.

Tipos de prueba:

- Pruebas unitarias
- Pruebas funcionales

3.4.2 Pruebas unitarias

Se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño del software. Tomando como guía la descripción del diseño al nivel de componentes se prueban importantes caminos de control para descubrir errores dentro de los límites del módulo (Presman, 2005).

Las pruebas unitarias se realizan con el método de caja blanca y la técnica del camino básico. El método del camino básico permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño procedural y usar esa medida como guía para la definición de un conjunto básico de caminos de

ejecución (Presman, 2005). Las unidades de prueba más pequeñas son las operaciones dentro de la clase.

A continuación se realiza la técnica del camino básico a la operación *MostrarsoftwaresInstalados()* que corresponde al requisito de prioridad alta *Mostrar listado de los softwares privativo.*

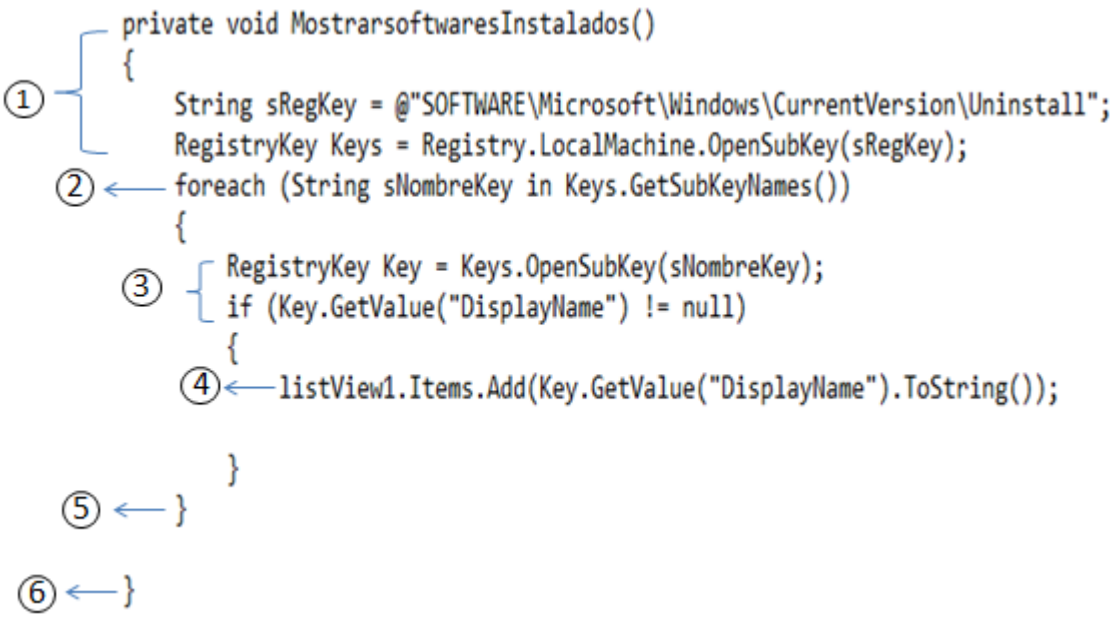


Figura 7: Código para las pruebas de unidad

Luego de numerar las líneas de código se diseña la gráfica del programa que describe el flujo de control lógico empleando nodos y aristas.

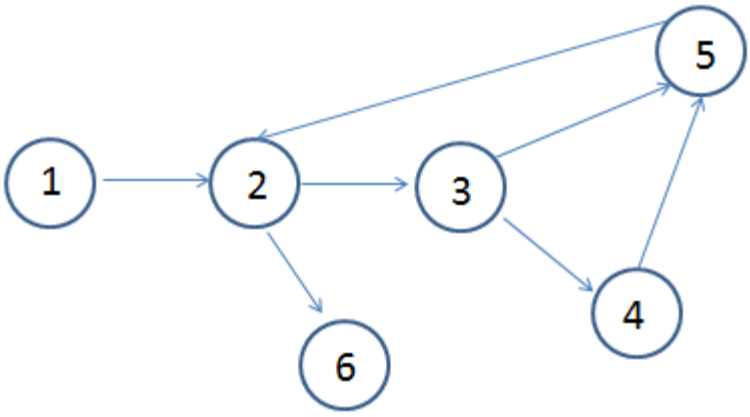


Figura 8: Grafo del flujo de control lógico

Una vez obtenido el grafo con 5 nodos y 6 aristas se procede a calcular la complejidad ciclomática V(G). La cual constituye una métrica de software que proporciona una medida cuantitativa de la complejidad lógica del programa (25). Existen varias formas de calcular este tipo de complejidad.

1. V(G)=(cantidad de aristas – cantidad de nodos) + 2 , V(G)=(6 – 5) + 2 = 3
2. Método del número de regiones, la cantidad de regiones encontradas son 3.
3. V(G)=nodos predicados + 1, 2 + 1 = 3

Como se puede observar por las tres variantes se obtuvo el mismo resultado.

Una ruta independiente es cualquier ruta del programa que ingrese al menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición. La cantidad de caminos es dada por la complejidad ciclomática por lo tanto se obtienen 3 caminos.

Tabla 7: Caminos de la complejidad ciclomática

No. Rutas	Caminos
1	1-2-6
2	1-2-3-5-6
3	1-2-3-4-5-6

Se diseñan casos de prueba a cada camino obtenido para establecer los posibles resultados esperados.

Tabla 8: Caso de Prueba de Unidad a la ruta 1

Caso de Prueba de Unidad	
No. Ruta : 1	**Ruta :**1-2-6
Nombre de la persona que realiza la prueba : Nelsón Yaidel Deus Soler	
Descripción de la prueba: La aplicación debe guardar en la cantidad de softwares instalados.	
Entrada: No aplica	
Resultado esperado: Se realiza la consulta al registro de *Windows* para obtener la cantidad de softwares instalados en la computadora, devuelve la cantidad de softwares.	
Evaluación de la prueba: Satisfactoria	

Tabla 9: Caso de Prueba de Unidad a la ruta 2

Caso de Prueba de Unidad	
No. Ruta : 2	**Ruta :**1-2-3-5
Nombre de la persona que realiza la prueba : Nelsón Yaidel Deus Soler	
Descripción de la prueba: La aplicación debe guardar la cantidad de softwares que deben estar instalados, además toma el primer nombre del software registrado.	
Entrada: No aplica	
Resultado esperado: Se realiza la consulta al registro de Windows para obtener la cantidad de softwares instalados en la computadora, devuelve la cantidad y el nombre del primer software registrado.	
Evaluación de la prueba: Satisfactoria	

Tabla 10: Caso de Prueba de Unidad a la ruta 3

Caso de Prueba de Unidad	
No. Ruta : 3	**Ruta :**1-2-3-4-5
Nombre de la persona que realiza la prueba : Nelsón Yaidel Deus Soler	
Descripción de la prueba: La aplicación muestra un listado de nombres de todos los softwares instalados en la computadora.	
Entrada: No aplica	
Resultado esperado: Se obtienen todos los softwares instalados en la máquina y el nombre de cada uno.	
Evaluación de la prueba: Satisfactoria	

3.4.3 Pruebas funcionales

Las pruebas funcionales no solo validan la transformación de una entrada en una salida, sino que también validan una característica completa. Estas son necesarias para que la aplicación funcione como un todo y determinar si el producto desarrollado cumple con las funcionalidades requeridas por el cliente (Presman, 2005).

A continuación, se describen los diseños de casos de prueba de los requisitos funcionales: mostrar listado de los softwares privativos y adicionar atributo de calidad.

Tabla 11: Caso de Prueba 1. Mostrar listado de los softwares privativos.

Escenario	Descripción	Respuesta del sistema	Flujo del sistema
EC 1.1Mostrar listado de software privativos.	La aplicación debe mostrar el listado de los softwares privativos.	La aplicación muestra el listado de los softwares privativos.	El especialista en migración de ejecutar la aplicación, acceder a la pestaña *Aplicaciones* y seleccionar *Buscar.*
EC 1.2 No mostrar listado de software privativo.	La aplicación no debe mostrar el listado de los softwares privativos.	La aplicación no muestra el listado de los softwares privativos.	

Tabla 12: Descripción de las variables

No	Nombre de las variables	Clasificación	Valor nulo	Descripción
1	Nombre	campo de texto	No	Este campo solo puede tener letras, con letra inicial mayúscula, la cantidad de caracteres máximo son de 50.
2	Tipo de dato	selección	No	Este campo debe seleccionar el tipo de dato string o entero.
3	Valor	Campo de texto	No	Este campo puede tener valores numéricos o letras. Admite como máximo 50 caracteres.

Tabla 13: Escenarios de las pruebas de validación

Escenario	Descripción	Nombre	Tipo de dato	Valor	Respuesta del sistema

Ec1.Adicionar atributo de forma correcta.	El sistema permite insertar el atributo en caso de que no exista ningún error.	Licencia	String	GPL	Adiciona el atributo de forma correcta
Ec2. Adicionar atributo de forma incorrecta.	El sistema no debe permitir datos no válidos.	54	String	56	No inserta los valores, muestra un mensaje de error.
Ec3. Adicionar atributo dejando campos en blanco.	Verifica si no existen campos en blanco.		String		No inserta los valores, muestra un mensaje de error.

3.4.4 Resultados de las pruebas

Los resultados obtenidos en las pruebas realizadas a la aplicación permitieron detectar un total de 15 no conformidades en 16 casos de pruebas realizados, donde 6 son de impacto alto, 7 de impacto medio y 2 de impacto bajo en un total de 3 iteraciones realizadas. Se utilizó el método de caja negra para realizar las pruebas sobre la interfaz de la aplicación. En la siguiente gráfica se ilustran las iteraciones realizadas y las no conformidades detectadas, las resueltas y las pendientes.

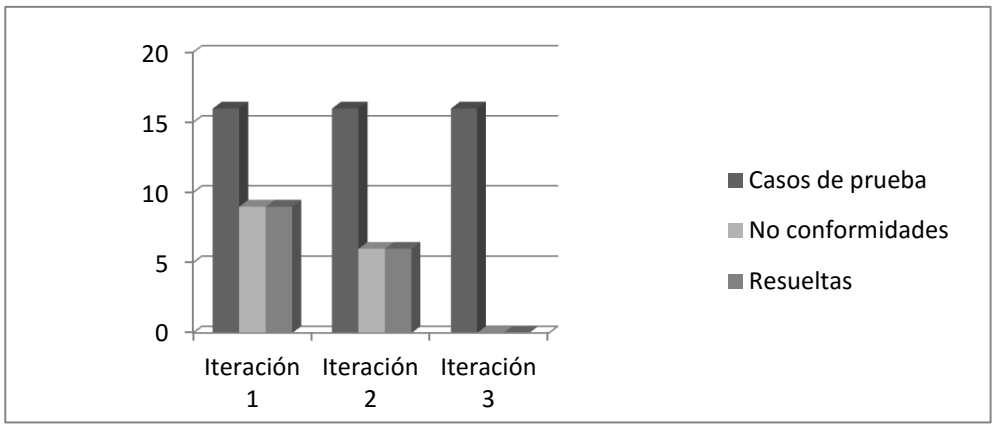


Figura 9: Resultados de las pruebas

Algunas de las no conformidades detectadas.

- En la HU_1 los softwares privativos mostrados se encuentran desordenados y con tipo de letra poco visible.
- En la HU_3 permite adicionar un atributo con valores vacíos.
- En la HU_3 no se muestran los valores de los atributos de la alternativa libre adicionada con anterioridad.

Conclusiones parciales

El empleo de los estándares de codificación definidos para la aplicación permitió desarrollar un código reutilizable. Mediante las pruebas realizadas se comprobó que las funcionalidades de la aplicación se ejecutan correctamente y se validó que la misma cumple con los requisitos del cliente. El diagrama de componentes permitió visualizar la estructura de la aplicación de escritorio y el comportamiento del servicio que estos componentes proporcionan, además mostró las dependencias entre las clases utilizadas y las librerías.

Conclusiones Generales

A partir de la investigación realizada y los resultados obtenidos se concluye lo siguiente:

- El estudio de las aplicaciones de inventario y de los directorios de software permitió concluir que estas no constituyen una solución para el proceso de migración en las estaciones de trabajo, por lo que se decide desarrollar una nueva aplicación que se ajuste a las necesidades de la migración a código abierto.

- El algoritmo propuesto permite la selección de las alternativas libres más adecuadas al software privativo, a través de atributos de calidad. Estos atributos deben ser evaluados constantemente ante posibles cambios en las aplicaciones libres, teniendo en cuenta los procesos clave de la institución donde se efectúa la migración a código abierto.

- Con la implementación de la propuesta de solución se obtuvo un sistema capaz de realizar la selección de las alternativas libres al software privativo, permitiendo agilizar los procesos de migración en las estaciones de trabajo.

- Los resultados de las pruebas unitarias y funcionales demostraron el correcto funcionamiento de la aplicación de escritorio para la selección de las alternativas libres al software privativo, logrando la satisfacción del cliente en cuanto al diseño y el manejo de las distintas funcionalidades.
- La aplicación de escritorio permite al especialista en migración mostrar el listado de los softwares privativos que se encuentran instalados en las computadoras de la institución durante el proceso de migración a código abierto y proponer la alternativa libre más adecuada, teniendo en cuenta atributos de calidad.

Recomendaciones

Al concluir la presente investigación, se recomienda:

- Agregar la funcionalidad de instalar y desinstalar las alternativas libres desde la aplicación.
- Permitir la actualización de la base de datos de las alternativas libres y aplicaciones privativas desde un servidor.
- Incorporar al algoritmo un mecanismo para evaluar todos los posibles valores de los atributos de calidad.
- Permitir en caso de que un software privativo no posea alternativa libre, que se muestren aplicaciones libres con las características y funcionalidades semejantes a esta.

Referencias Bibliográficas

Ambler, Scott W. 2010. *Introduction to UML 2 Package Diagrams .* 2010.

Creately.com. creately.com. [Online] 2017. [Cited: marzo 14, 2017.]

directory.fsf.org. 2016. *directory.fsf.org.* [Online] 2016. [Cited: Noviembre 29, 2016.] directory.fsf.org.

foreui.com. *www.foreui.com.* [Online] 2015. [Cited: Diciembre 4, 2016.] www.foreui.com.

freealts.com. *www.freealts.com.* [Online] [Cited: Noviembre 15, 2016.] www.freealts.com.

FusionInventory. 2016. *fusioninventory.org.* [Online] 2016. [Cited: Diciembre 5, 2016.] fusioninventory.org. 6.

Gamma, Eric, et al. *Design Patterns - Elements of Reusable Object-Oriented Software. .*

glpi-project.org. *www.glpi-project.org.* [Online] 2016. [Cited: Noviembre 16, 2016.] glpi-project.org/spip.php?article13.

GNU. *www.gnu.org.* [Online] 2014. [Cited: Febrero 12, 2017.] www.gnu.org/philosophy/categories.es.html.

González Carrera, Maria Leysi, Pérez Villazón, Yoandy and González Hernández, Cesar. 2016. *Atributos de calidad para la evaluación y selección de aplicaciones.* La habana : s.n., 2016.

Larman, Craig. 2005. *UML y patrones, 2da Edición.* 2005.

Microsoft. 2016. *msdn.microsoft.com.* [Online] 2016. [Cited: Diciembre 2016, 2016.] msdn.microsoft.com/es-es/library/aa287558(v=vs.71).aspx.

—. 2015. *msdn.microsoft.com.* [Online] 2015. [Cited: Diciembre 21, 2016.] msdn.microsoft.com/es-es/library/fx6bk1f4(v=vs.100).aspx.

ocsinventory. 2016. *www.ocsinventory-ng.org.* [Online] 2016. [Cited: Noviembre 5, 2016.] www.ocsinventory-ng.org.

omg.org. 2016. *www.omg.org.* [Online] 2016. [Cited: Dicembre 10, 2016.] www.omg.org.

open-audit.org. 2017. *www.open-audit.org.* [Online] 2017. [Cited: Noviembre 24, 2016.] www.open-audit.org.

osalt.com. *www.osalt.com.* [Online] 2015. [Cited: Diciembre 14, 2016.] www.osalt.com/.

Referencias Bibliográficas

Patricio , LeterierTorres. 2003. *Metodologías ágiles en el desarrollo del software.* 2003.

Pérez Guevara, Diosbel, et al. 2014. *Plataforma Cubana de Migración a Código Abierto.* La Habana : s.n., 2014.

Presman, Roger. 2005. *Ingenieria del software.Un enfoque práctico 6taEdicion.* 2005.

Rodríguez Sanchez, Tamara. 2015. *Metodología de desarrollo para la actividad productiva de la UCI.* 2015.

Software, Laboratorio Nacional de Calidad del. 2009. www.academia.edu. [Online] 2009.

software.com. 2016. *www.software.com.* [Online] 2016. [Cited: Diciembre 20, 2016.] www.software.com.ar/p/visual-paradigm-para-uml.

sqlite.org. 2016. *www.sqlite.org.* [Online] 2016. [Cited: diciembre 10, 2016.] www.sqlite.org.

St. Laurent, Andrew M. 2008. *Understanding Open Source and Free Software Licensing.* 2008.

Stallman, Richard Matthew. 2004. *Software libre para una sociedad libre.* 2004.

um.es. 2014. *www.um.es.* [Online] 2014. [Cited: Octubre 24, 2016.] www.um.es/docencia/barzana/IAGP/Iagp2.html.

Villazón, Yoandy Pérez, et al. 2014. *Buenas Prácticas para la Migración Código Abierto.* 2014.

Anexos

Tabla 14: HU Modificar atributo

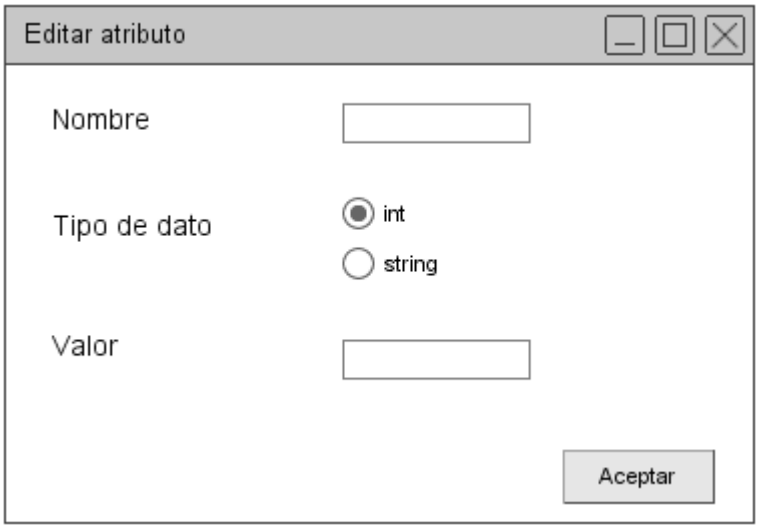
Historia de usuario	
Número: 5	**Nombre del requisito:** Modificar atributo
Programador: Nelson Yaidel Deus Soler	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 100 horas
Riesgo en Desarrollo: No se encuentra la información correspondiente a la alternativa libre.	**Tiempo Real:** 100 horas
Descripción: El sistema permite modificar un atributo para realizar la selección de las alternativas libres al software privativo.	
Observaciones: Los especialistas de migración pueden modificar los valores de los atributos de calidad.	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 15: HU Actualizar la información de las alternativas libres en la base de datos

Historia de usuario	
Número: 6	**Nombre del requisito:** Actualizar la información de las alternativas libres en la base de datos
Programador: Nelson Yaidel Deus Soler	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 120 horas
Riesgo en Desarrollo: No contenga alternativas disponibles para actualizar	**Tiempo Real:** 120 horas
Descripción: La base de datos con los softwares libres se actualizará de forma manual en la base de datos de la aplicación.	
Prototipo elemental de interfaz gráfica de usuario:	

Tabla 16: HU Mostrar listado de atributos de calidad

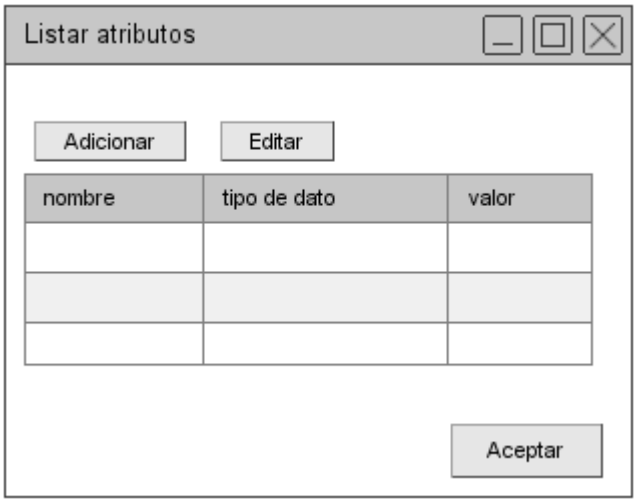
Historia de usuario	
Número: 7	**Nombre del requisito:** Mostrar listado de atributos
Programador: Nelson Yaidel Deus Soler	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 100 horas
Riesgo en Desarrollo: No exista atributos	**Tiempo Real:** 100 horas
Descripción: Después de insertar varios atributos el especialista puede observar el listado de todos ellos.	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 17: HU Insertar software libre

Historia de usuario	
Número: 8	**Nombre del requisito**: Insertar *software* libre
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 120 horas**
Riesgo en Desarrollo:	**Tiempo Real: 120 horas**
Descripción: La aplicación permite adicionar un *software* libre para ser propuesto como alternativa. Además del nombre, descripción y categoría debe tener en cuenta el valor de cada atributo insertado.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario:	


Adicionar SWL
Nombre
Descripción
Categoría
Terminal
Insertar

Tabla 18: HU Eliminar software libre

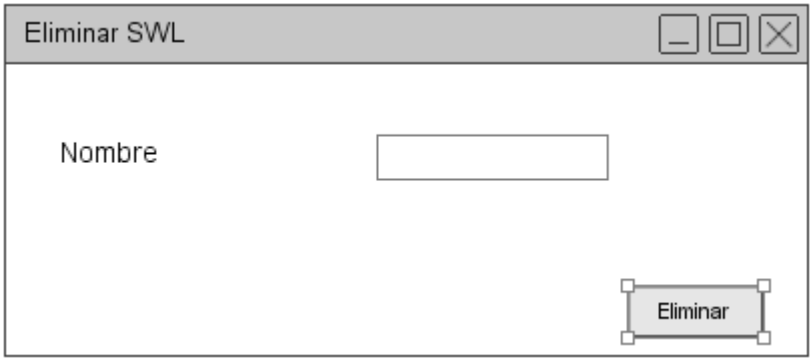
Historia de usuario	
Número: 9	**Nombre del requisito**: Eliminar *software* libre
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 120 horas**
Riesgo en Desarrollo:	**Tiempo Real: 120 horas**
Descripción: La aplicación debe permitir eliminar un software libre previamente insertado en la base de datos.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 19: HU Eliminar software libre

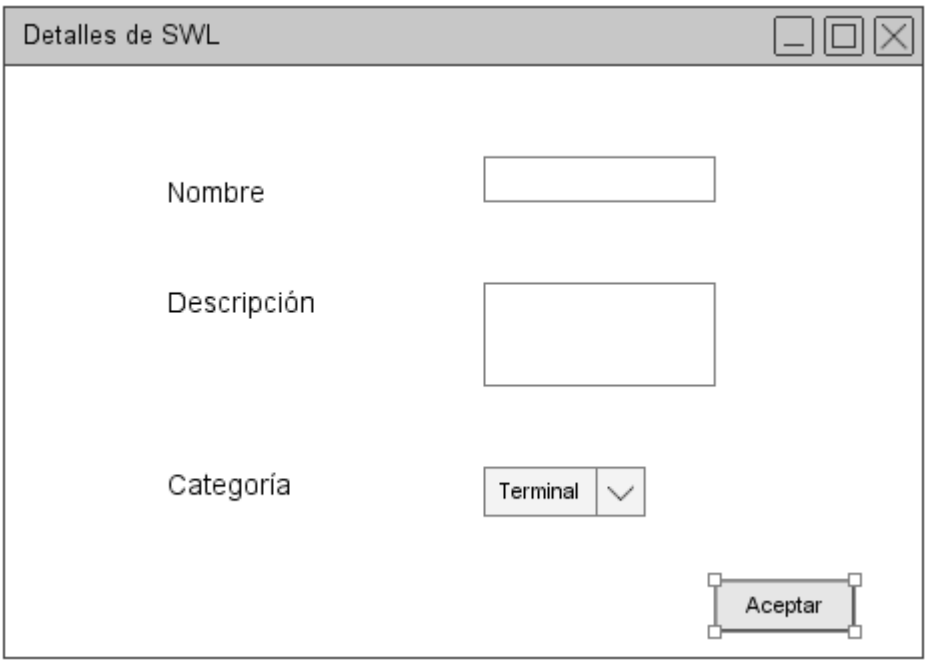
Historia de usuario	
Número: 10	**Nombre del requisito:** Detalles del software libre
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 120 horas**
Riesgo en Desarrollo:	**Tiempo Real: 120 horas**
Descripción: La aplicación debe permitir mostrar los detalles de cada software libre.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 20: HU Listado de software libre

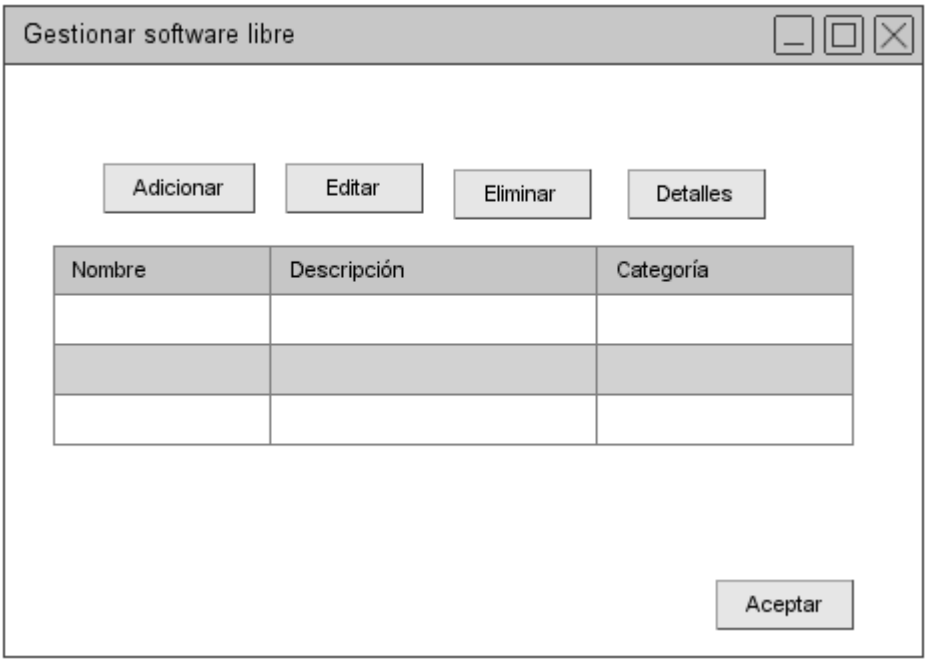
Historia de usuario	
Número: 11	**Nombre del requisito**: Listado de *software* libre
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 120 horas**
Riesgo en Desarrollo:	**Tiempo Real: 120 horas**
Descripción: Se muestran todo el *software* libre que tiene la base de datos.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 21: HU Insertar software privativo

Historia de usuario	
Número: 12	**Nombre del requisito**: Insertar *software* privativo
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 80 horas**
Riesgo en Desarrollo:	**Tiempo Real: 80 horas**
Descripción: La aplicación permite adicionar un *software* privativo.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 22: HU Editar software privativo

Historia de usuario	
Número: 13	**Nombre del requisito**: Editar *software* privativo
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 100 horas**
Riesgo en Desarrollo:	**Tiempo Real: 120 horas**
Descripción: La aplicación permite modificar los valores de cada *software* privativo.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 23: HU Listar los *softwares* privativos

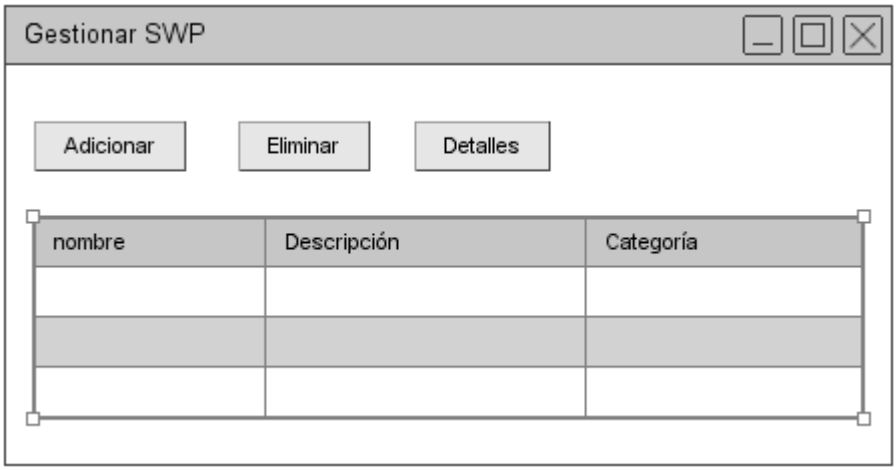
Historia de usuario	
Número: 14	**Nombre del requisito**: Listar *software* privativo
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 100 horas**
Riesgo en Desarrollo:	**Tiempo Real: 120 horas**
Descripción: La aplicación permite mostrar los softwares libres de la base de datos.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 24: HU Eliminar software privativo

Historia de usuario	
Número: 15	**Nombre del requisito**: Eliminar *software* privativo
Programador: Nelsón Yaidel Deus Soler	**Iteración Asignada: 1**
Prioridad: Media	**Tiempo Estimado: 90 horas**
Riesgo en Desarrollo:	**Tiempo Real: 90 horas**
Descripción: La aplicación permite eliminar un software privativo que se encuentre en la base de datos.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario: