Universidad de las Ciencias Informáticas

Módulo PackageManager para la Suite de Administración para Servidores de Entornos Productivo.

Trabajo de Diploma

Autor:

Yunior Hernández Rodríguez

Tutor:

Ing. Michel Miranda Cairo

Declaración de Autoría

Declaro que soy el único autor de este trabajo que lleva por título "Módulo PackageManager para la Suite de Administración para Servidores de Entornos Productivo" y autorizo a la Universidad de las Ciencias Informáticas (UCI) para que haga el uso que estime pertinente con este trabajo.

Para que así conste firmo la presente a los _____ días del mes de _____ de 2012.

Autor:

_____

Yunior Hernández Rodríguez

Tutor:

_____

Ing. Michel Miranda Cairo

Agradecimientos

A mi familia por su apoyo constante. A mis compañeros de estudio que durante estos años han sido mi familia. A los amigos incondicionales que siempre han estado a mi lado. A mi abuela Erodys y mi abuelo Alberto por todo el amor que me han dado. Al 8203 por ser el mejor grupo de la Universidad de las Ciencias Informáticas. A mi tutor por su confianza. A todos los que de una forma u otra han aportado su grano de arena para la construcción de este sueño.

Dedicatoria

A mí Dios, pues cada logro en mi vida es más suyo que mío.

A mí madre que con sudor, lágrima y sangre ha forjado mi camino.

A mí hermano Abraham por motivarme a ser su ejemplo.

A mí novia por ser mi gran amiga y mi gran amor.

A mí familia por confiar en mí.

A mís amigos por ser los únicos hermanos que he tenido la oportunidad de elegir.

Resumen

Se propone desarrollar el módulo PackageManager que se integre a la “Suite para la Administración de Servidores de Entornos Productivos” (SASEP) el cual debe permitir la gestión de paquetes de forma remota y mediante una interfaz visual. Se decide hacer uso de C++ como lenguaje para el desarrollo por la flexibilidad que ofrece el mismo y su total compatibilidad con sistemas Linux. Se selecciona a QT como framework pues el mismo permite un desarrollo cómodo con C++ y provee al desarrollador un gran número de potentes librerías que permiten la agilización del trabajo. Se opta por la metodología Programación Extrema para el desarrollo de la aplicación pues la misma se centra en el desarrollo del software y se adapta a los cambios de requisitos entendiéndolos como algo natural. Se realiza el análisis y diseño de la aplicación y posteriormente se lleva a cabo la implementación de esta. Finalmente se somete a prueba el software resultante y se ofrecen las conclusiones de la investigación realizada.

Índice

Introducción ____ ***8***

Capítulo 1 Fundamentación Teórica ____ ***12***

1.1 Introducción ____ **12**

1.2 Proceso de Gestión de Software. ____ **12**

1.2.1 Instalación. ____ 12

1.2.2 Desinstalación. ____ 13

1.2.3 Repositorios. ____ 13

1.2.4 Gestores de Paquetes. ____ 15

1.3 Principales herramientas para la Gestión de Software. ____ **16**

1.3.1 APT ____ 16

1.3.2 Aptitude ____ 16

1.3.3 YUM ____ 17

1.3.4 Synaptic ____ 17

1.3.5Adept ____ 18

1.3.6 KPackage ____ 18

1.3.7 Ubuntu Software Center ____ 19

1.3.8 RPM ____ 19

1.3.9 Summon ____ 19

1.4 Protocolo SSH ____ **19**

1.5 Tendencias y Tecnologías Actuales ____ **20**

1.5.1 Herramientas ____ 20

1.5.1.1 Suite Administrativa para Servidores de Entornos Productivos, SASEP ____ 21

1.5.1.2 NetBeans ____ 22

1.5.1.3 Eclipse ____ 22

1.5.1.4 CodeBlocks ____ 23

1.5.1.5 KDevelop ____ 23

1.5.1.6 Qt creator ____ 24

1.5.3 Metodologías de Desarrollo ____ 24

1.5.3.1 RUP ____ 25

1.5.3.2 SCRUM ____ 25

1.5.3.3 Programación Extrema (XP) ____ 26

1.5 Conclusiones del Capítulo ____ **27**

Capítulo 2: Análisis y diseño del sistema 29

2.1 Introducción. 29

2.2 Descripción de los procesos vinculados al campo de acción. 29

2.2.1 Propuesta del sistema 29

2.3 Funcionalidades del sistema. 30

2.3.1 Funcionalidades. 30

2.4 Características del sistema. 30

2.4.1 Características 31

2.5 Personal relacionado con la aplicación. 31

2.6 Historias de usuario. 31

2.7 Planificación de las Historias de Usuario. 34

2.8 Diseño 35

2.8.1 Tarjetas CRC 35

2.9 Conclusiones. 38

Capítulo 3: Implementación y Prueba 39

3.1 Introducción. 39

3.2 Iteraciones. 39

3.2.1 Primera iteración 39

3.2.2 Segunda iteración 42

3.2.3 Tercera iteración 43

3.3 Prueba. 46

3.3.1 Prueba de aceptación 47

3.3.1.2 Iteración 1 47

3.3.1.2 Iteración 2 49

3.3.1.3 Iteración 3 50

3.4 Conclusiones 52

Conclusiones. 54

Bibliografía 55

Introducción

En los últimos años se ha producido una explosión masiva relacionada con las Tecnologías de la Información y la Comunicaciones (TICs). Las bondades de las TICs cada vez son más asequibles y accesibles para todos y no solo para un sector privilegiado de la sociedad aunque el control de las mismas sí está reservado para un grupo elitista. Internet pasó de ser un instrumento especializado de la comunidad científica a ser una red de fácil uso y accesibilidad para todos. La informática y las comunicaciones se han beneficiado de la miniaturización de los componentes lo que ha permitido la reducción de los precios de ventas de estos productos.

Inicialmente, debido al alto costo de producción de los equipos informáticos, el software era gratuito o el precio del mismo venía incluido en el hardware. En la actualidad esta filosofía de mercado se ha visto afectada por el desarrollo de las TICs, siendo la tendencia que disminuya el precio del hardware y que el software sea la parte más importante del gasto en tecnología informática.

En nuestro país se viene incursionando en el campo de la informática y las comunicaciones desde inicios de 1959. Dentro de los logros fundamentales destacan la confección de la computadora CID-201 en 1970 y la creación de los Joven Club de Computación en 1987. No es hasta el año 2002 que se le presta especial atención a la producción de software a escala industrial y es en este año que se decide crear la Universidad de las Ciencias Informáticas (UCI). Ya en el año 2004 dicho centro comienza a tener una presencia productiva en la industria del software.

La Universidad de las Ciencias Informáticas además de ser una casa de altos estudios es el principal productor de software de nuestro país. En ella existen entornos laborales destinados a la producción. Cada entorno productivo puede contar con uno o más servidores sobre los cuales se alojan las aplicaciones desarrolladas y las herramientas necesarias para el buen desempeño del proyecto. También se pueden almacenar en estos servidores la información relacionada con el trabajo que se está desempeñando.

Los servidores usualmente se instalan sobre ordenadores que tienen como sistema operativo alguna versión de GNU/Linux. Este sistema operativo se caracteriza fundamentalmente por ser un software no privativo y por su robustez. Su principal desventaja es que demanda un amplio conocimiento por parte de los gestores de configuración. El correcto funcionamiento de un servidor depende en gran medida de

que el mismo cuente con el software indicado y que el mismo se encuentre correctamente configurado. Esto exige que los administradores tengan que conocer un gran número de comandos, de parámetros de configuración y además deban conocer en totalidad cuáles son los paquetes a instalar y las dependencias de los mismos. De ahí que el **problema de la investigación** sea el siguiente:

¿Cómo facilitar la gestión de paquetes en los servidores de los entornos productivos de la UCI?

El problema planteado se enmarca en el **objeto de estudio**:

Proceso de gestión de paquetes en servidores.

Para darle solución al problema de la investigación se define como **objetivo general**:

Desarrollar el módulo PackageManager que se integre a la "Suite para la Administración de Servidores de Entornos Productivos" (SASEP) el cual debe permitir la gestión de paquetes de forma remota y mediante una interfaz visual en los entornos de desarrollo de la UCI.

Se tiene como **campo de acción:**

Proceso de gestión de paquetes en los servidores de los entornos productivos en la UCI.

La **idea a defender** es la siguiente:

La creación de un módulo para SASEP que permita la gestión de paquetes de forma visual y remota, contribuirá a facilitar el trabajo de los gestores de configuración de la UCI.

Para darle cumplimiento al objetivo general se trazaron los siguientes **objetivos específicos**:

- Analizar los elementos teóricos conceptuales sobre la gestión de paquetes.
- Seleccionar la metodología adecuada para el desarrollo de la aplicación.
- Diseñar un módulo para SASEP que permita la gestión de los paquetes en un servidor de forma visual y remota.
- Implementar y probar la herramienta creada.

Para darle cumplimiento a los objetivos específicos se definen las siguientes **tareas de la investigación:**

1. Estudio del marco teórico y estado del arte de los gestores de paquetes.
2. Análisis sobre metodologías y herramientas para el desarrollo del módulo PackageManager.
3. Identificación de las principales característica, buenas prácticas y políticas de seguridad de servicios.
4. Descripción detallada del funcionamiento del servidor.
5. Elaboración del modelo de análisis y diseño basado en la metodología escogida.
6. Implementación de la solución.
7. Realización de pruebas para comprobar el correcto funcionamiento de las funcionalidades desarrolladas.

Métodos teóricos

Analítico-Sintético: Para el análisis de teorías, documentos, y materiales relacionados con la gestión de paquetes sobre sistemas operativos de forma que luego de procesar la información se pueda abordar a conclusiones.

Histórico-Lógico: Para el estudio de la evolución y desarrollo de la configuración de servidores.

Métodos empíricos

Entrevistas: Para obtener información de los administradores y gestores sobre el trabajo que realizan.

Estructura del Trabajo

Capítulo 1. Fundamentación Teórica:

Se observan los conceptos fundamentales referentes a la gestión de paquetes en servidores de entornos productivos. Se hace un estudio de los principales gestores de paquetes y se establecen sus ventajas y desventajas. Se realiza un breve estudio de los lenguajes de programación candidatos para la implementación de la aplicación con el objetivo de definir el que será seleccionado para el desarrollo. Se describen las tecnologías y herramientas a utilizar para dar solución al problema.

Capítulo 2. Análisis y diseño del sistema:

Se describe la propuesta de sistema. Se especifican las funcionalidades del sistema y las características del mismo. Se identifican y describen las historias de usuario. Se identifica el personal relacionado con la aplicación y se establece el diseño mediante el uso de tarjetas CRC (clases, responsabilidad, colaboración).

Capítulo 3. Implementación y prueba:

Se describe el proceso de implementación de la herramienta definiendo para cada iteración las tareas por Historias de Usuario (HU) correspondientes. Se exponen los resultados arrojados por las pruebas realizadas a las diferentes funcionalidades implementadas.

Capítulo 1 Fundamentación Teórica

1.1 Introducción

En el presente capítulo se ofrece una descripción detallada del proceso de gestión de software en diversos sistemas operativos. Se hace especial énfasis en los sistemas Linux, fundamentalmente algunas versiones derivadas de Debian. Se ilustra cómo se realiza la instalación y desinstalación de paquetes en los mismos y los problemas que se pueden presentar durante el proceso. Se presenta un estudio del arte relacionado con los sistemas gestores de paquetes en la actualidad, sus características y funcionalidades más relevantes y se pone a disposición del lector un estudio detallado de las herramientas, metodologías y tecnologías posibles a utilizar.

1.2 Proceso de Gestión de Software.

Este epígrafe tiene como tema central la gestión del software, fundamentalmente en los sistemas Linux. Se explicará en qué consisten los procesos de instalación y desinstalación y se abordarán los conceptos de repositorios y gestores de paquetes.

1.2.1 Instalación.

Según la Real Academia Española en su Diccionario de la Lengua Española, vigésima segunda edición, instalar se define como: “Poner o colocar en el lugar debido a alguien o algo” (1). La empresa de telecomunicaciones EMOPA plantea que: “El término instalación se relaciona con el acto de instalar, que supone colocar, arreglar o disponer determinados elementos para que funcionen o que cumplan ciertos objetivos. La instalación es algo que se establece en pos de lograr determinada función u objetivo” (2).

La instalación de software se definirá en este trabajo como “la acción de transferir el mismo al ordenador y configurarlo para ser usado con el objetivo que se le creó”. Mientras más complejo sea el software, mayor sea el volumen de archivos que contenga el mismo o más interdependiente sea con otro software, el riesgo de la ocurrencia de fallas durante el proceso de instalación del mismo será más alto. Es por este motivo que el desarrollo de un proceso de instalación confiable y seguro es un aspecto sumamente importante.

En la actualidad se han desarrollado nuevas normas y técnicas con el objetivo de estandarizar el proceso de instalación de software y las mismas son cada vez más potentes. Las técnicas básicas para la instalación de software son las siguientes:

1- Los archivos se copian en algún lugar del directorio.
2- Se instala en el ordenador un software, el cual en la mayoría de los casos viene implícito en la propia instalación del sistema operativo y este se encarga del proceso de instalación en el ordenador.
3- El sistema operativo o algún programa se encargan de instalar los paquetes de software con todas las dependencias requeridas. A esta aplicación se le conoce como Sistema Gestor de Paquetes.

1.2.2 Desinstalación.

La Real Academia Española en su Diccionario de la Lengua Española, vigésima tercera edición, define desinstalar como: “Eliminar del disco duro de un ordenador o computador los archivos necesarios para el funcionamiento de un programa” (3).

La desinstalación de un software consiste en revertir todas las modificaciones realizadas por este sobre el sistema operativo. Para lograr una correcta desinstalación no solo deben ser borrados los archivos de la raíz, sino que también se deben deshacer cambios en otros aspectos como pueden ser:

- Eliminar usuarios que hayan sido creados por concepto de instalación del software.
- Retirar consecuentemente permisos.
- Borrar directorios.
- Llevar un registro o *log* de los cambios realizados.

Según el artículo “Desinstalación de Paquetes” publicado en el sitio de Geofísica de la Universidad Nacional Autónoma de México (UNAM), se plantea que: “La desinstalación de un paquete elimina los ficheros de la aplicación y los asociados a ella de su máquina. Cuando un paquete es desinstalado, todos los ficheros que usa y no son necesitados por otros paquetes del sistema son también borrados" (4).

1.2.3 Repositorios.

El Centro Nacional de Electromagnetismo Aplicado de Cuba plantea que: “Un repositorio, depósito o archivo, es un sitio web centralizado donde se almacena y

mantiene información digital, habitualmente bases de datos o archivos informáticos, siendo éstos de fácil acceso y uso por los usuarios" (5). En Linux estos repositorios se usan de forma intensiva, pues los mismos sirven como fuente de software para el sistema. Normalmente el software se centraliza y se organiza en los repositorios con el objetivo de ofrecerle a los usuarios un sencillo control sobre los paquetes disponibles en los mismos, permitiéndole a estos instalar y desinstalar software fácilmente.

Un repositorio se puede encontrar publicado en un servidor ya sea ftp (sigla en inglés de File Transfer Protocol - Protocolo de Transferencia de Archivos) o http (sigla en inglés de Hypertext Transfer Protocol- Protocolo de Transferencia de Hipertexto), o en el disco local la estación de trabajo. En ellos se mantienen los paquetes actualizados con sus últimas versiones. Los programas son verificados por el personal responsable del desarrollo de la distribución lo que garantiza que no exista problema por incompatibilidad.

En Ubuntu (distribución para la cual se propone el presente trabajo) las aplicaciones de los repositorios se dividen en cuatro secciones fundamentales para diferenciar la prioridad que les otorga los desarrolladores de la aplicación. Estos componentes son:

- **main**: Contiene los paquetes que cumplen los requisitos de la licencia de Ubuntu, y para los que hay soporte disponible por parte de su equipo. Está pensado para que incluya todo lo necesario para la mayoría de los sistemas Linux. Los paquetes de este componente poseen ayuda técnica garantizada y mejoras de seguridad.
- **restricted**: Contiene los programas soportado por los desarrolladores de Ubuntu debido a su importancia, pero que no está disponible bajo ningún tipo de licencia libre para incluir en main. En este lugar se incluyen los paquetes tales como los controladores propietarios de algunas tarjetas gráficas, como por ejemplo, los de ATI y NVIDIA. Se limita en algunos casos el acceso al código fuente, lo cual reduce los niveles de ayuda.
- **universe**: Contiene una amplia gama de programas, que pueden o no tener una licencia restringida, pero que no recibe apoyo por parte del equipo de Ubuntu sino por parte de la comunidad. Esto permite que los usuarios instalen toda clase de programas en el sistema guardándolos en un lugar aparte de los paquetes soportados: main y restricted.
- **multiverse**: Contiene los paquetes sin soporte debido a que no cumplen los requisitos de aplicaciones de código libre.

1.2.4 Gestores de Paquetes.

Luego de conocer en qué consiste el proceso de Gestión de Software y qué son los Repositorios, se hace necesario conocer el término Gestores de Paquetes o Gestor de Paquetes.

Según el profesor asociado de la Universidad de Atenas Diomidis Spinellis, un sistema gestor de paquetes es un software que: "…simplifica la instalación y mantenimiento de software mediante la estandarización y la organización de la producción y el consumo de las colecciones de software…" (6). En el sitio oficial de openSUSE, una de las distribuciones de Linux de más prestigio, se definen los sistemas gestores de paquetes como: "Una colección de herramientas que proporciona un método consistente de instalación, actualización y eliminación de software en su sistema" (7).

Teniendo en cuenta las definiciones anteriores se puede concluir que un Gestor de Paquetes es un software conformado por una selección de herramientas que facilitan y automatizan el proceso de instalación, configuración, actualización y desinstalación de paquetes en un sistema operativo.

Normalmente en los sistemas Linux el software se encuentra almacenado en forma de paquetes, casi siempre encapsulados en un solo fichero. Diomidis Spinellis plantea que: "Los paquete contienen en un formato estandarizado la fuente de un módulo de software o de código compilado (o el lugar donde estos se alojan), junto con su documentación y los metadatos" (6). Como plantea Spinellis, estos paquetes no solo son portadores del software sino que cuentan también con una serie de información referente al propio software como puede ser el nombre completo del mismo, la versión, su distribuidor, una descripción del mismo, las dependencias, entre otras. Un paquete está conformado por una serie de programas que se distribuyen en forma conjunta debido a que el funcionamiento de uno de ellos se complementa con los otros o depende de ellos.

Como su nombre lo indica, los gestores de paquetes gestionan y organizan todos los paquetes que se encuentran instalados en nuestro sistema y mantienen su usabilidad. Esto se logra gracias a la combinación de las siguientes técnicas:

- Comprobación de la suma de verificación para evitar que haya diferencias entre la versión local de un paquete y la versión oficial.
- Comprobación de la firma digital.
- Instalación, actualización y eliminación simple de paquetes.

- Resolución de dependencias para garantizar que el software funcione correctamente.
- Búsqueda de actualizaciones para proveer la última versión de un paquete, ya que normalmente solucionan bugs y proporcionan actualizaciones de seguridad.
- Agrupamiento de paquetes según su función para evitar la confusión al instalarlos o mantenerlos.

1.3 Principales herramientas para la Gestión de Software.

En la actualidad con la constante evolución de los sistemas Linux y con la variedad de distribuciones existentes, ha surgido un gran número de gestores de paquetes. En el presente epígrafe se ofrece un estudio de los gestores más utilizados y de un referente creado en nuestro centro de altos estudios.

1.3.1 APT

APT o Advanced Packaging Tool (Herramienta Avanzada de Empaquetado) es un gestor de paquetes creado por el proyecto Debian. Apt tiene a su favor el hecho de simplificar considerablemente el proceso de gestión de paquetes en Linux.

Según sus desarrolladores: “APT es un sistema de gestión de paquetes de software. Dispone de varias interfaces para la gestión de paquetes, tales como Aptitude para la línea de órdenes o Synaptic para el sistema de ventanas de X. Algunas opciones sólo están implementadas en apt-get” (8).

APT no es un programa sino una colección de bibliotecas de funciones de C++ y es utilizado por gran número de programas de líneas de comandos para la gestión de software.

Inicialmente APT solo permitía el trabajo con paquetes .deb, en Debian y en distribuciones derivadas de este. En la actualidad ha sido modificado para el trabajo con paquetes .rpm y para ser compatible con otros sistemas operativos como Mac OS y OpenSolaris.

1.3.2 Aptitude

Daniel Burrows creador de Aptitude lo define como: “Una interfaz de texto para el sistema de paquetes de Debian GNU/Linux que permite al usuario ver la lista de

paquetes y realizar tareas de gestión tales como instalar, actualizar o eliminar paquetes. Puede llevar a cabo las acciones con una interfaz gráfica o en la línea de órdenes" (9).

Aptitude es una interfaz para *APT,* creada en 1999 y basada en una biblioteca *ncurses*, mediante la cual incorpora algunos elementos comunes a otras interfaces gráficas, como son los menús desplegables. Se creó para experimentar con un diseño más orientado a objetos, con el objetivo de que resultara un programa flexible y con características extensibles. Muestra una lista de paquetes de software y permite al usuario elegir de modo interactivo cuáles desea instalar o eliminar. Dispone de un poderoso sistema de búsqueda que facilita al usuario entender las complejas relaciones de dependencia que puedan existir entre los paquetes.

En el año 2000, se reescribió toda la interfaz gráfica creándose una nueva arquitectura, basada en la librería *libsigc++* y en conceptos de modernos *toolkits* de controles, como GTK+ y Qt. Este cambio permitió que la interfaz ganara en usabilidad, con nuevas características como los menús desplegables y cajas de diálogo emergentes.

1.3.3 YUM

YUM (Yellow dog Updater, Modified) al decir de Robert G. Brown "… es una herramienta diseñada para proporcionar a los usuarios o administradores de sistemas, la capacidad de automatizar completamente todos los aspectos de Red Hat Package Manager (RPM) tales como instalación de paquetes y administración de sistemas basados en RPM…" (10). YUM es una utilidad para líneas de comandos aunque existen otras aplicaciones que le ofrecen al mismo una interfaz gráfica de usuario como pup, pirut y yumex.

YUM se reescribió totalmente a partir de su predecesora YUP (Yellowdog Updater). Esta herramienta se confeccionó con el objetivo de gestionar los paquetes de los sistemas Red Hat. Actualmente existen otras distribuciones que hacen uso de esta aplicación como Fedora, CentOS y otras basadas en RPM.

1.3.4 Synaptic

Según sus propios desarrolladores "**Synaptic** es un programa de gestión de paquetes gráfico para apt. Ofrece las mismas características que la utilidad de línea de

comandos apt-get con una interfaz gráfica de usuario front-end basado en GTK +" (11).

Synaptic es uno de los gestores de paquetes más utilizados en distribuciones Linux. El mismo se usa fundamentalmente en sistemas basados en paquetes .deb, aunque también puede ser usado en sistemas basados en RPM.

Synaptic permite gestionar los paquetes del repositorio mediante un menú interactivo y sencillo. Mediante un clic el usuario puede actualizar su sistema, listar los paquetes que se pueden instalar al igual que los ya instalados. Posee un avanzado filtro de búsquedas, permite reparar los problemas de dependencias ocurridos, así como deshacer y rehacer las últimas selecciones de paquetes.

Kynaptic es un gestor de software que utiliza el código fuente base del Synaptic y posee las mismas funcionalidades que este en entornos de escritorio KDE.

1.3.5Adept

Adept es una interfaz gráfica de APT semejante a Synaptic pero desarrollado utilizando las librerías QT, por lo que se usa fundamentalmente en entornos de escritorio KDE.

Está compuesto por varios componentes que realizan funciones específicas:

- adept_updater: asistente de actualizaciones para el sistema.
- adept_batch: script para automatizar tareas mediante Adept.
- adept_manager: administrador de paquetes (similar a Synaptic).
- adept_installer: administrador de aplicaciones (añadir/eliminar programas).
- adept_notifier: notificador de actualizaciones disponibles.

1.3.6 KPackage

KPackage es un administrador de paquetes de KDE. Soporta BSD, Debian, Gentoo, RPM y paquetes Slackware. Provee Interfaz gráfica de usuario para administrar y actualizar paquetes existentes así como para instalar y obtener nuevos paquetes. Adicionalmente provee funcionalidad para ayudar a manejar la cache (memoria en la que se almacenan una serie de datos para su rápido acceso) de los paquetes.

1.3.7 Ubuntu Software Center

Según Andrew Higginson **Ubuntu Software Center** se define en su manual de ayuda como: “Una interfaz gráfica para la administración de paquetes en Ubuntu” (12).

Ubuntu Software Center es un software que hace uso de las librerías GTK+ para su interfaz gráfica de usuario y se utiliza para la gestión de paquetes en Ubuntu. El mismo está elaborado para el manejo de paquetes .deb y facilita en gran medida el proceso de gestión de paquetes ofreciéndole al usuario una interfaz intuitiva y organizada donde encontrar todo el software que se encuentra en los repositorios. Este gestor permite instalar, desinstalar, comprar, manejar repositorios, y actualizar aplicaciones de una manera simple y unificada con una sola aplicación general.

1.3.8 RPM

RPM es un sistema de gestión de paquetes originalmente desarrollado por *Red Hat* para la distribución Red Hat Linux. Los paquetes utilizados por el sistema tienen extensión *.rpm*, formato que actualmente es la base de la Base Estándar de Linux (LSB) y se emplean en diversas distribuciones Linux así como en otros sistemas operativos como Novell Net-Ware e IBM's AIX.

1.3.9 Summon

Summon es un gestor de paquetes desarrollado en la Universidad de las Ciencias Informáticas. Su objetivo fue permitirles a los usuarios del sistema operativo Nova, cuando el mismo estaba inspirado en Gentoo, la gestión de los programas que estos deseaban utilizar en sus ordenadores. Posee una interfaz gráfica sencilla que se integra sin inconveniente alguno con Gnome, entorno predefinido de la distribución.

1.4 Protocolo SSH

SSH (sigla en inglés de Secure SHELL - Intérprete de órdenes segura) es el nombre por el cual se conoce a un protocolo y el programa que lo implementa. El mismo sirve para acceder a máquinas remotas a través de una red y manejar por completo las mismas mediante el uso de un intérprete de comandos. SSH también permite copiar datos de forma segura, gestionar claves RSA y transferir datos de cualquier aplicación por un canal seguro.

Según Varsity Drive Raleigh en su “Red Hat Enterprise Linux 4: Manual de referencia” SSH se define como: “…un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación remota tales como FTP o Telnet, SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas…”. (13)

El protocolo SSH proporciona los siguientes tipos de protección:

- Después de la conexión inicial, el cliente puede verificar que se está conectando al mismo servidor al que se conectó anteriormente.
- El cliente transmite su información de autenticación al servidor usando una encriptación robusta de 128 bits.
- Todos los datos enviados y recibidos durante la sesión se transfieren por medio de encriptación de 128 bits, esto los hace extremamente difícil de descifrar y leer.
- El cliente tiene la posibilidad de reenviar aplicaciones X11(X Window System: Protocolo que permite la interacción gráfica en red entre un usuario y una o más computadoras) desde el servidor. Esta técnica, llamada *reenvío por X11*, proporciona un medio seguro para usar aplicaciones gráficas sobre una red.

1.5 Tendencias y Tecnologías Actuales

En el desarrollo de todo software la selección de las herramientas, lenguaje y metodologías a utilizar juega un papel importante garantizando un óptimo desempeño del sistema.

1.5.1 Herramientas

En la actualidad con el desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs), la elección de una herramienta adecuada para el cumplimiento de un determinado propósito puede volverse complejo. Por este motivo se hace necesario un estudio de las tendencias y las herramientas actuales más usadas con el fin de seleccionar las que mejores se acoplen a las necesidades que exige el desarrollo de la solución propuesta.

1.5.1.1 Suite Administrativa para Servidores de Entornos Productivos, SASEP

SASEP es una herramienta que permite la administración remota y visual de servidores desarrollada en el Departamento de Implantación y Soporte Técnico del Centro de Tecnologías para la Formación en la UCI. Está basada en una arquitectura modular extensible a través de plugins (módulo de software que se añade a otro con el objetivo de extender sus funcionalidades), los cuales posibilitan adicionarle nuevas funcionalidades para la administración de diferentes servicios.

Toda la comunicación de SASEP con el servidor es tramitada vía SSH, lo que la convierte en una conexión con un nivel excelente en cuanto a seguridad. Así mismo proporciona una interfaz, como se muestra en la figura 1, que debe ser implementada por los plugins que se desarrollen.

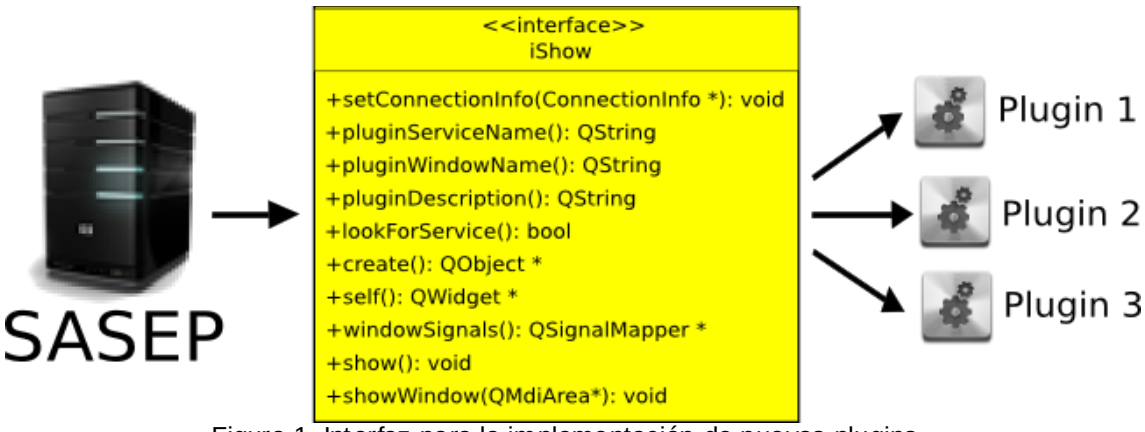

Figura 1. Interfaz para la implementación de nuevos plugins.

Al estar desarrollada en el lenguaje C++ y el framework Qt 4.7, es necesario para un correcto desempeño que los plugins sean implementados siguiendo estas pautas. Por tal motivo en esta investigación no se analizan otras tecnologías.

C++ es un lenguaje orientado a objetos. El mismo se deriva de C. En realidad es un superconjunto de C, que nació para añadirle cualidades y características de las que carecía. El resultado es que al igual que su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se le han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción.

Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario creada por la compañía noruega Trolltech y utilizada en KDE, un entorno de escritorio

para sistemas GNU/Linux y BSD entre otros. Utiliza el lenguaje de programación C++ de forma nativa aunque existen enlaces soportados para C, Python (PyQt), Java (Qt Jambi), Perl (PerlQt), Gambas (gb.qt), Ruby (QtRuby), PHP (PHP-Qt) y Mono (Qyoto). La API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML y para el manejo de ficheros y además de estructuras de datos tradicionales.

1.5.1.2 NetBeans

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo como es el NetBeans C/C++ Pack que permite el desarrollo con C y C++ sobre este IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

NetBeans es un proyecto de código abierto de mucho éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000. Actualmente este proyecto pertenece a Oracle.

1.5.1.3 Eclipse

Según el sitio oficial de ayuda de Eclipse, en la guía de programadores, este se define como una plataforma diseñada para la construcción de herramientas y el desarrollo de aplicaciones. En cuanto al diseño, la plataforma no proporciona una gran cantidad de funcionalidades al usuario final por si mismo. Su valor fundamental es que alienta el rápido desarrollo de funciones integradas sobre la base de un modelo plugin (14)

Eclipse proporciona una interfaz de usuario (UI) común para trabajar con herramientas. Está diseñado para ejecutarse en múltiples sistemas operativos al mismo tiempo que ofrece una integración sólida con cada sistema operativo subyacente. Debido a su arquitectura basada en plugins se puede desarrollar sobre Eclipse usando diversos lenguajes, frameworks y librerías. En el caso de Qt, Eclipse cuenta con el “Qt Eclipse Integration for C++” el cual es un plugin que al decir de sus desarrolladores permite a los programadores crear, construir, depurar y ejecutar aplicaciones Qt desde el IDE (15).

1.5.1.4 CodeBlocks

A decir de sus desarrolladores CodeBlocks es: "… un IDE para C++ libre, diseñado para satisfacer las necesidades más exigentes de sus usuarios… muy extensible y totalmente configurable…" (16). Está basado en la plataforma de interfaces gráficas WxWidgets, por lo que puede usarse libremente en diversos sistemas operativos, y está licenciado bajo la Licencia pública general de GNU.

"Es un IDE construido como un núcleo altamente expansible mediante complementos (plugins). Actualmente la mayor parte de la funcionalidad viene provista por los complementos incluidos predeterminadamente. No es un IDE autónomo que acepta complementos, sino que es un núcleo abstracto donde los complementos se convierten en una parte vital del sistema. Esto lo convierte en una plataforma muy dinámica y potente, no solo por la facilidad con que puede incluirse una nueva funcionalidad, sino por la capacidad de poder usarla para construir otras herramientas de desarrollo tan solo añadiendo complementos." (17).

1.5.1.5 KDevelop

Según el sitio oficial de KDevelop este se define como: "Un moderno entorno de desarrollo integrado (IDE) para C++ (y otros lenguajes)" (18). A diferencia de otros entornos de desarrollo, KDevelop no posee un compilador propio. Debido a esto depende de gcc para producir código binario.

Este IDE usa por defecto el editor de texto Kate. Dentro de sus principales características se pueden encontrar las siguientes:

- Editor de código fuente con destacado de sintaxis e indentado automático (Kate).
- Gestión de diferentes tipos de proyectos, como CMake, *Automake*, *qmake* (para proyectos basados en la biblioteca Qt y *Ant* (para proyectos basados en Java).
- Navegador entre clases de la aplicación.
- Front-end para gcc, el conjunto de compiladores de GNU.
- Front-end para el depurador de GNU.
- Asistentes para generar y actualizar las definiciones de las clases y el framework de la aplicación.
- Completado automático del código en C y C++.
- Compatibilidad nativa con Doxygen.

- Permite control de versiones.

1.5.1.6 Qt creator

En palabras de sus creadores Qt Creator "…es un entorno multiplataforma de desarrollo integrado (IDE) adaptado a las necesidades de los desarrolladores de Qt. Este se ejecuta en Windows, Linux/X11 y Mac OS X y permite a los desarrolladores crear aplicaciones para múltiples escritorios y plataformas de dispositivos móviles" (19).

Dentro de sus principales características tenemos:

- Posee un avanzado editor de código C++
- Soporta los lenguajes: Python y PySide, Ada, Pascal, Perl, PHP y Ruby.
- Posee también una GUI integrada y diseñador de formularios.
- Herramienta para proyectos y administración.
- Ayuda sensible al contexto integrado.
- Depurador visual.
- Resaltado y auto-completado de código.
- Soporte para refactorización de código.

1.5.3 Metodologías de Desarrollo

Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software. Se puede definir como un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software. Se debe destacar que el éxito del producto depende en gran parte de la metodología seleccionada por el equipo de desarrollo. En la actualidad existen dos tipos de metodologías de desarrollo: Las Metodologías Tradicionales o Pesadas y las Metodologías Ágiles o Livianas las cuales al decir de Pressman "…buscan la satisfacción del cliente y la entrega temprana de software incremental; equipos de proyectos pequeños y con alta motivación; métodos informales; un mínimo de productos de trabajo de la ingeniería del software y una simplicidad general del desarrollo…" (20). Cualquiera que sea la metodología seleccionada, debe ser capaz de maximizar el potencial del equipo de desarrollo, además de aumentar la calidad del producto con los recursos y tiempos establecidos.

1.5.3.1 RUP

El Rational Unified Process o Proceso Unificado de Desarrollo. Es un proceso de ingeniería de software que suministra un enfoque para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga la necesidad del usuario final dentro de un tiempo y presupuesto previsible. Es una metodología de desarrollo iterativo enfocada hacia los casos de uso, manejo de riesgos y el manejo de la arquitectura.

En la obra **El Proceso Unificado de Desarrollo de Software** de Booch, Jacobson y Rumbaugh se plantea que: "…el Proceso Unificado más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos" (21).

El RUP mejora la productividad del equipo ya que permite que cada miembro del grupo sin importar su responsabilidad específica acceda a la misma base de datos de conocimiento. Esto hace que todos compartan el mismo lenguaje, la misma visión y el mismo proceso acerca de cómo desarrollar software.

Las características principales que presenta RUP son:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo)
- Pretende implementar las mejores prácticas en Ingeniería de Software
- Desarrollo iterativo
- Administración de requisitos
- Uso de arquitectura basada en componentes
- Control de cambios
- Modelado visual del software
- Verificación de la calidad del software

1.5.3.2 SCRUM

"SCRUM es un marco simple para la colaboración eficaz de los equipos en proyectos complejos. Proporciona un pequeño conjunto de reglas que crean simplemente la estructura suficiente para que los equipos sean capaz de enfocar su innovación en la solución de lo que podría ser un desafío insuperable" (22).

SCRUM, más que una metodología de desarrollo de software, es una forma de auto-gestión de los equipos de programadores. Un grupo de programadores deciden cómo hacer sus tareas y cuánto van a tardar en ello. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro.

SCRUM permite además seguir de forma clara el avance de las tareas a realizar, de forma que los "jefes" puedan ver día a día cómo progresa el trabajo.

Sin embargo, no es una metodología de desarrollo, puesto que no indica qué se debe hacer para hacer el código. Debería, por tanto, complementarse con alguna otra metodología de desarrollo. Se usa frecuentemente con las metodologías ágiles y en concreto, con la programación extrema.

1.5.3.3 Programación Extrema (XP)

En palabras de los desarrolladores de XP: “Extreme Programming es exitoso porque hace hincapié en la satisfacción del cliente. En vez de entregar todo lo que pueda desear en una fecha lejana en el futuro este proceso ofrece el software que necesita como usted lo necesite. Extreme Programming permite a los desarrolladores responder con confianza a las necesidades cambiantes de los clientes, incluso al final del ciclo de vida” (23).

La programación extrema es un enfoque de la ingeniería de software formulado por Kent Beck y constituye la más destacada de las metodologías ágiles de desarrollo de software. Se diferencia de las metodologías tradicionales principalmente en que hace más énfasis en la adaptabilidad que en la previsibilidad. Considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Promueve que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación más realista que intentar definir todos los requisitos al comienzo e invertir esfuerzos después en controlar los cambios en los mismos. Se puede considerar la programación extrema como la adopción de las mejores prácticas de desarrollo de software y su aplicación de manera dinámica durante el ciclo de vida del software.

Las características fundamentales de la metodología son:

- Desarrollo iterativo e incremental.

- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto.
- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto.
- Simplicidad en el código: Es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo en caso necesario, que realizar algo complicado y quizás nunca utilizarlo.

1.5 Conclusiones del Capítulo

Después de haber realizado el estudio de los principales gestores de paquetes, se puede concluir que los mismos no dan solución al problema de la investigación por las siguientes razones:

1- Ninguno de los gestores estudiados permiten la gestión remota de paquetes.
2- La única solución factible que podemos obtener de alguno de ellos, como son los gestores de líneas de comandos tales como el APT, Aptitude o RPM, es la de establecer una conexión mediante SSH con el servidor a administrar y haciendo uso de líneas de comandos, realizar la gestión de los paquetes.
3- Los gestores de paquetes que ofrecen al usuario una interfaz visual con el objetivo de facilitar el trabajo tales como Synaptic, Kynaptic o Adept, no permiten la gestión remota de paquetes.

Por estas razones se decide la creación de un plugin para la herramienta SASEP que permita la gestión remota de paquetes en los servidores de entornos productivos de la

UCI. Luego de un detallado estudio de las tendencias y tecnologías actuales se decide hacer uso de C++ como lenguaje para la implementación por la flexibilidad que ofrece el mismo y su total compatibilidad con sistemas Linux. Se selecciona a QT como framework pues el mismo permite un desarrollo cómodo con C++ y provee al programador un gran número de potentes librerías que permiten la agilización del trabajo. Además C++ es el lenguaje con el cual se desarrolló la herramienta SASEP mediante el uso del framework Qt 4.7.

Se elige a Qt Creator como IDE de desarrollo pues es el IDE por excelencia para el trabajo con el framework Qt, existe cierta experiencia en su uso y se cuenta con una amplia documentación en español e inglés. Finalmente se decidió optar por la metodología Programación Extrema para el desarrollo de la aplicación pues la misma se centra en el desarrollo del software y se adapta a los cambios de requisitos entendiéndolos como algo natural. Además, adopta las mejores prácticas de desarrollo de software y permite la aplicación de estas de forma dinámica.

Capítulo 2: Análisis y diseño del sistema

2.1 Introducción.

Si bien el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en fases, por lo que el presente capítulo se centra fundamentalmente en las tres primeras fases de XP propuestas por José Joskowicz en su libro “Reglas y Prácticas en eXtreme Programming” (24). Además se ofrece la propuesta de solución para el desarrollo de una herramienta capaz de gestionar los paquetes en los servidores de entornos productivos de forma remota y mediante una interfaz visual.

2.2 Descripción de los procesos vinculados al campo de acción.

Actualmente en la Universidad de las Ciencias Informáticas la gestión de paquetes en entornos productivos se realiza haciendo uso de una consola mediante líneas de comandos. Por este motivo se propone la creación de un plugin que se integre a la Suite para la Administración de Servidores en Entornos Productivos (SASEP). Este debe permitir la gestión de paquetes de forma remota ofreciéndoles una interfaz visual sencilla e intuitiva a los gestores de configuración.

2.2.1 Propuesta del sistema

Tras el análisis concerniente a la configuración de servidores en entornos productivos de la Universidad de las Ciencias Informáticas, específicamente el proceso de gestión de software, podemos concluir que se hace necesario la creación de una herramienta que permita la gestión de software de forma remota y visual por lo que se propone la creación de un módulo que integre la herramienta SASEP.

La creación del módulo para la gestión de paquetes permitirá, mediante una interfaz visual, gestionar los paquetes de un servidor. Para lograr este propósito, la herramienta debe ser capaz de establecer una conexión vía SSH con el servidor deseado. Este ofrecerá una interfaz visual que servirá como front-end del gestor de líneas de comandos APT. Esto le permitirá al gestor de configuración realizar todo el proceso de gestión de software con simples clic y siendo transparente para el mismo toda la serie de comandos generados.

Esta aplicación debe ser capaz de permitirle al administrador instalar y desinstalar un paquete determinado. Permitirá también listar los paquetes existentes en los repositorios, mostrar la información relacionada con los mismos, actualizar el sistema operativo, gestionar el listado de los repositorios.

2.3 Funcionalidades del sistema (FS).

Una funcionalidad es una especificación de qué se debería implementar. Es una descripción de cómo se debe comportar el sistema, de un atributo o una propiedad. Una condición o capacidad que debe cumplir o poseer un sistema o componente de un sistema.

2.3.1 Funcionalidades.

Las funcionalidades con las cuales debe cumplir la aplicación a desarrollar son:

FS1: Establecer conexión a servidor remoto mediante SSH.

FS2: Listar paquetes existentes en los repositorios.

FS3: Listar paquetes instalados en el servidor.

FS4: Instalar nuevos paquetes en el servidor.

FS5: Desinstalar paquetes en el servidor.

FS6: Actualizar el sistema operativo.

FS7: Agregar nuevos repositorios.

FS8: Eliminar repositorios.

FS9: Editar repositorios.

FS10: Mostrar información de los paquetes seleccionados.

FS11: Hacer búsqueda de paquetes.

2.4 Características del sistema.

Las características del sistema definen aspectos que sin ser funcionalidades resultan deseables desde el punto de vista del usuario. Describen aspectos del sistema que

son visibles por el usuario que no incluyen una relación directa con el comportamiento funcional del sistema.

2.4.1 Características

Las características con los cuales debe cumplir la aplicación son:

- **Usabilidad:** La aplicación contará con una interfaz fácil e intuitiva que facilitará la gestión de paquetes.
- **Portabilidad:** La aplicación se ejecutará sobre Ubuntu y Nova.
- **Seguridad:** El flujo de información y el intercambio de claves se realizará mediante el protocolo seguro SSH.
- **Software:** El sistema debe ser desarrollado usando C++ como lenguaje. Se debe utilizar como framework para el desarrollo del mismo QT. Se desarrollará sobre el sistema operativo Ubuntu.

2.5 Personal relacionado con la aplicación.

El personal relacionado con la aplicación son todas aquellas personas que intervienen en el desarrollo de la misma o harán uso de esta.

Personal relacionado	Justificación
Desarrollador	Es la persona que intervendrá directamente en el proceso de implementación de la aplicación.
Gestores de configuración	Son aquellas personas que harán uso de la aplicación para la gestión de paquetes en los servidores de los entornos productivos de la Universidad de las Ciencias Informáticas.

Tabla 1. Descripción del personal relacionado con el sistema.

2.6 Historias de usuario.

Las Historias de Usuario (HU) son una descripción de las necesidades funcionales. Según Daniel H. Steinberg and Daniel W. Palmer en su libro *Extreme Software Engineering*, las Historias de Usuario son “una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos” (25). Las mismas permiten responder rápidamente a los requisitos cambiantes siendo esta una de sus principales ventajas.

Historia de Usuario						Número:	1
Nombre:	Establecer conexión SSH						
Prioridad	Alta	**Complejidad**:	Alta	**Estimación**:	1	**Iteración**:	1
Descripción:							
Establecer una conexión segura con la estación servidor que permita el flujo de la información.							
Información Adicional:							
Da cumplimiento al requisito FS1.							

Tabla 2: Descripción de la Historia de Usuario: Establecer conexión SSH

Historia de Usuario						Número:	2
Nombre:	Listar paquetes						
Prioridad	Alta	**Complejidad**:	Alta	**Estimación**:	2	**Iteración**:	1
Descripción:							
Listar los paquetes existentes en los repositorios y los instalados.							
Información Adicional:							
Da cumplimiento a los requisitos FS2, FS3.							

Tabla 3: Descripción de la Historia de Usuario: Listar paquetes.

Historia de Usuario						Número:	3
Nombre:	Instalar paquetes						
Prioridad	Alta	**Complejidad**:	Alta	**Estimación**:	2	**Iteración**:	1
Descripción:							
Instalar paquetes en el servidor desde los repositorios correspondientes.							
Información Adicional:							
Da cumplimiento a los requisitos FS4.							

Tabla 4: Descripción de la Historia de Usuario: Instalar paquetes.

Historia de Usuario		Número:	4
Nombre:	Desinstalar paquetes		

Prioridad	Alta	**Complejidad**:	Alta	**Estimación**:	2	**Iteración**:	2
Descripción:							
Desinstalar cualquier paquete en el servidor.							
Información Adicional:							
Da cumplimiento al requisito FS5.							

Tabla 5: Descripción de la Historia de Usuario: Desinstalar paquetes.

Historia de Usuario						**Número:**	**5**
Nombre:	Actualizar el sistema operativo.						
Prioridad	Media	**Complejidad**:	Baja	**Estimación**:	1	**Iteración**:	2
Descripción:							
Hacer una actualización completa del sistema operativo con las últimas actualizaciones de software disponibles en los repositorios.							
Información Adicional:							
Da cumplimiento al requisito FS6.							

Tabla 6: Descripción de la Historia de Usuario: Actualizar el sistema operativo.

Historia de Usuario						**Número:**	**7**
Nombre:	Gestionar repositorios						
Prioridad	Media	**Complejidad**:	Baja	**Estimación**:	1	**Iteración**:	3
Descripción:							
Adicionar y eliminar repositorios.							
Información Adicional:							
Da cumplimiento a los requisitos FS7, FS8 y FS9.							

Tabla 7: Descripción de la Historia de Usuario: Gestionar repositorios.

Historia de Usuario						**Número:**	**8**
Nombre:	Mostrar información de los paquetes.						
Prioridad	Baja	**Complejidad**:	Media	**Estimación**:	1	**Iteración**:	3

Descripción:
Muestra la información referente a los paquetes seleccionados.
Información Adicional:
Da cumplimiento al requisito FS10.

Tabla 8: Descripción de la Historia de Usuario: Mostrar información de los paquetes.

Historia de Usuario						**Número:**	**9**
Nombre:	Buscar paquetes.						
Prioridad	Baja	**Complejidad**:	Alta	**Estimación**:	1	**Iteración**:	3
Descripción:							
Permite realizar una búsqueda de un paquete determinado.							
Información Adicional:							
Da cumplimiento al requisito FS11.							

Tabla 9: Descripción de la Historia de Usuario: Buscar paquetes.

2.7 Planificación de las Historias de Usuario.

Iteración	Historias de Usuarios	Tiempo
1	1. Establecer conexión SSH 2. Listar paquetes 3. Instalar paquetes	5 semanas
2	4. Desinstalar paquetes 5. Actualizar el sistema operativo	4 semanas
3	6. Gestionar repositorios 7. Mostrar información de los paquetes. 8. Buscar paquetes.	3 semanas

Tabla 10: Distribución de Historias de Usuario por iteración.

2.8 Diseño

Según Manuel Calero Solís en el **V Encuentro usuarios xBase 2003 MADRID:** "El diseño crea una estructura que organiza la lógica del sistema, un buen diseño permite que el sistema crezca con cambios en un solo lugar" (26) . Según Joskowicz "la metodología XP hace especial énfasis en los diseños simples y claros" (27).

XP a diferencia de la mayoría de las metodologías que usan diagramas para el desarrollo de modelos, hace uso de tarjetas para la representación de las clases.

2.8.1 Tarjetas CRC

Las tarjetas utilizadas por XP para la representación de las clases son llamadas CRC (clases, responsabilidad, colaboración). Son sencillas de utilizar y entender y permiten al equipo de trabajo en su totalidad participar en el diseño del sistema. La estructura de estas tarjetas es la siguiente:

Tarjeta CRC	
Clase: Nombre de la clase que se está modelando.	
Súper Clase: Nombre de la clase padre en la herencia.	
Sub Clase(s): Nombre de la(s) clase(s) hija en la herencia.	
Responsabilidades: Es una descripción de alto nivel del propósito de la clase.	**Colaboraciones:** Indica con cuáles otras clases se requiere relación para cumplir la responsabilidad.

Tabla 11: Descripción de una tarjeta CRC.

Tarjeta CRC 1
Clase: gestorMain

Súper Clase:	
Sub Clase(s):	
Responsabilidades: Es la clase principal y permite la gestión de todos los procesos en la aplicación.	**Colaboraciones:** • Connection. • Information. • repoManager.

Tabla 12: Tarjeta CRC 1.

Tarjeta CRC 2	
Clase: Connection	
Súper Clase:	
Sub Clase(s):	
Responsabilidades: Es la responsable de establecer la conexión con el servidor y de escribir en consola y leer la salida de la misma.	**Colaboraciones:** .

Tabla 13: Tarjeta CRC 2.

Tarjeta CRC 3
Clase: Information
Súper Clase:
Sub Clase(s):

Responsabilidades: Muestra la salida arrojada por algunos procesos.	**Colaboraciones:** • Connection.

Tabla 14: Tarjeta CRC 3.

Tarjeta CRC 4	
Clase: repoManager	
Súper Clase:	
Sub Clase(s):	
Responsabilidades: Se encarga de la gestión de los repositorios.	**Colaboraciones:** • Connection. • addRepo. • editRepo.

Tabla 15: Tarjeta CRC 4.

Tarjeta CRC 5
Clase: addRepo
Súper Clase:
Sub Clase(s):

Responsabilidades: Se encarga de adicionar nuevos repositorios.	**Colaboraciones:** • Connection.

Tabla 16: Tarjeta CRC 5.

Tarjeta CRC 6	
Clase: editRepo	
Súper Clase:	
Sub Clase(s):	
Responsabilidades: Se encarga de editar los repositorios.	**Colaboraciones:** • Connection.

Tabla 17: Tarjeta CRC 6.

2.9 Conclusiones.

En el presente capítulo se realizó una descripción de la solución propuesta. Se definieron y redactaron las historias de usuario así como las tareas de la ingeniería correspondientes a cada una de ellas. Todo esto arrojó un tiempo de 12 semanas para el cumplimiento de las tareas planteadas y por consiguiente la elaboración del módulo.

Capítulo 3: Implementación y Prueba

3.1 Introducción.

Luego del diseño de la aplicación obtenido en el capítulo anterior se procede a dar inicio a la fase de iteraciones. La metodología XP plantea que la implementación debe realizarse de forma iterativa e incremental, obteniéndose en cada iteración un producto que debe ser probado y mostrado al cliente, logrando así una constante retroalimentación entre el cliente y los desarrolladores. En el presente capítulo también se exponen los resultados arrojados por las pruebas realizadas a las diferentes funcionalidades implementadas.

3.2 Iteraciones.

Durante esta fase se vinculan las Historias de Usuarios con tareas concretas de desarrollo, las cuales se le encomiendan a cada programador el cual debe asumir la responsabilidad de la implementación de las mismas. Para el desarrollo total de la aplicación se definieron tres iteraciones de forma tal que al concluir cada iteración se pudiera obtener un producto funcional que cumpliera con las características deseadas por el cliente y el cual se pudiera someter a las pruebas correspondientes.

Para la asignación de los puntos de estimación se tuvo en cuenta que un punto equivale a una semana la cual cuenta con cinco días de trabajo y cada día cuenta con ocho horas laborales.

3.2.1 Primera iteración

En esta iteración se implementan las historias de usuario 1,2 y 3. Al concluir esta iteración se debe haber implementado todas las funcionalidades propuestas de forma tal que se le pueda mostrar el resultado al cliente mediante un producto funcional.

Tareas por Historia de Usuario definidas en la primera iteración.

Tarea	
Número de tarea: 1	**Número de HU:** 1

Nombre de la tarea: Establecer conexión SSH	
Tipo de tarea: Desarrollo	**Puntos de estimación: 1**
Fecha de Inicio: 30/1/2012	**Fecha de Fin: 6/2/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: Se establece conexión con el servidor mediante el protocolo seguro SSH.	

Tabla 18: Tarea: Establecer conexión SSH.

Tarea	
Número de tarea: 2	**Número de HU:** 2
Nombre de la tarea: Listar paquetes de los repositorios	
Tipo de tarea: Desarrollo	**Puntos de estimación: 1**
Fecha de Inicio: 8/2/2012	**Fecha de Fin: 15/2/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: Se muestran al usuario todos los paquetes existentes en el repositorio.	

Tabla 19: Tarea: Listar paquetes de los repositorios.

Tarea	
Número de tarea: 3	**Número de HU:** 2
Nombre de la tarea: Listar paquetes instalados en el servidor.	
Tipo de tarea: Desarrollo	**Puntos de estimación: 2**
Fecha de Inicio: 21/2/2012	**Fecha de Fin: 28/2/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: Se muestran al usuario todos los paquetes instalados en el servidor.	

Tabla 20: Tarea: Listar paquetes instalados en el servidor.

Tarea	
Número de tarea: 4	**Número de HU:** 3
Nombre de la tarea: Instalar paquetes en el servidor	
Tipo de tarea: Desarrollo	**Puntos de estimación: 2**
Fecha de Inicio: 2/3/2012	**Fecha de Fin: 16/3/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: El usuario puede instalar paquetes en un servidor de forma remota.	

Tabla 21: Tarea: Instalar paquetes.

3.2.2 Segunda iteración

En esta iteración se implementarán las historias de usuarios 4 y 5. Al concluir la misma se debe obtener una segunda versión de la aplicación con nuevas funcionalidades listas para mostrar al usuario. También al finalizar esta iteración se debe contar con un 70% aproximadamente del total del módulo.

Tareas por Historia de Usuario definidas en la segunda iteración.

Tarea	
Número de tarea: 5	**Número de HU:** 4
Nombre de la tarea: Desinstalar paquetes del servidor	
Tipo de tarea: Desarrollo	**Puntos de estimación: 2**
Fecha de Inicio: 19/3/2012	**Fecha de Fin: 2/4/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: El usuario puede desinstalar los paquetes del servidor de forma remota.	

Tabla 22: Tarea: Desinstalar paquetes del servidor.

Tarea	
Número de tarea: 6	**Número de HU:** 5
Nombre de la tarea: Actualizar el sistema operativo	
Tipo de tarea: Desarrollo	**Puntos de estimación: 2**

Fecha de Inicio: 3/4/2012	**Fecha de Fin: 10/4/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: El usuario puede actualizar el sistema operativo.	

Tabla 23: Tarea: Actualizar el sistema operativo.

3.2.3 Tercera iteración

En esta iteración se implementarán las historias de usuario 6,7 y 8. Al concluir esta iteración se le debe ofrecer al usuario un producto final totalmente funcional que satisfaga las necesidades del mismo.

Tareas por Historia de Usuario definidas en la tercera iteración.

Tarea	
Número de tarea: 7	**Número de HU:** 6
Nombre de la tarea: Agregar nuevos repositorios	
Tipo de tarea: Desarrollo	**Puntos de estimación: 0.3**
Fecha de Inicio: 23/4/2012	**Fecha de Fin: 35/4/2012**
Programador Responsable: Yunior Hernández Rodríguez	

Descripción: El usuario puede agregar nuevos repositorios al listado de fuentes del sistema operativo.

Tabla 24: Tarea: Agregar nuevos repositorios.

Tarea	
Número de tarea: 8	**Número de HU:** 6
Nombre de la tarea: Eliminar repositorios	
Tipo de tarea: Desarrollo	**Puntos de estimación: 0.3**
Fecha de Inicio: 25/4/2012	**Fecha de Fin: 27/4/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: El usuario puede eliminar repositorios del listado de fuentes del sistema operativo.	

Tabla 25: Tarea: Eliminar repositorios.

Tarea	
Número de tarea: 9	**Número de HU:** 6
Nombre de la tarea: Editar repositorios	
Tipo de tarea: Desarrollo	**Puntos de estimación: 0.4**

Fecha de Inicio: 27/4/2012	**Fecha de Fin: 30/4/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: El usuario puede editar los repositorios del listado de fuentes del sistema operativo.	

Tabla 26: Tarea: Editar repositorios.

Tarea	
Número de tarea: 10	**Número de HU:** 7
Nombre de la tarea: Mostrar información de los paquetes seleccionados	
Tipo de tarea: Desarrollo	**Puntos de estimación: 1**
Fecha de Inicio: 2/5/2012	**Fecha de Fin: 9/5/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: Se le muestra al usuario la información de los paquetes que selecciona.	

Tabla 27: Tarea: Mostrar información de los paquetes seleccionados.

Tarea	
Número de tarea: 11	**Número de HU:** 8

Nombre de la tarea: Buscar paquetes	
Tipo de tarea: Desarrollo	**Puntos de estimación: 1**
Fecha de Inicio: 10/5/2012	**Fecha de Fin:17/5/2012**
Programador Responsable: Yunior Hernández Rodríguez	
Descripción: Se busca un paquete determinado.	

Tabla 28: Tarea: Buscar paquete.

3.3 Prueba.

Según William Hetzel en su libro **The Complete Guide to Software Testing "**La prueba de software es una actividad dirigida a la evaluación de un atributo o capacidad de un programa o sistema y la determinación de que cumple con los resultados requeridos**"** (28)**.** En palabras de Jiantao Pan en su artículo **Software Testing "**El propósito de las pruebas puede ser la garantía de calidad, verificación y validación, o estimación de la fiabilidad**"** (29). Teniendo en cuenta estas opiniones se puede concluir que las pruebas evalúan el producto garantizando que el mismo cumpla con la calidad y fiabilidad requerida.

La metodología XP plantea que las pruebas se deben dividir en dos grupos fundamentales:

Pruebas unitarias: El propósito de estas pruebas es aislar cada parte del programa y mostrar que las partes individuales son correctas. Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad.

Pruebas de aceptación: "Las pruebas de aceptación son consideradas como pruebas de caja negra ("*Black box system tests*"). Los clientes son responsables de verificar

que los resultados de estas pruebas sean correctos. Asimismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución" (27).

3.3.1 Prueba de aceptación

Se considera que las pruebas de aceptación poseen un peso superior a las unitarias, pues las mismas arrojan a vista de los desarrolladores la satisfacción del cliente. Por este motivo se puede decir que estas ponen fin a una iteración dando inicio a la siguiente.

Durante las iteraciones, las Historias de Usuario correspondientes se expresarán en pruebas de aceptación. El cliente desde su punto de vista creara los casos de prueba en conjunto con los desarrolladores para probar que una HU ha sido implementada correctamente. Por cada HU se elaborarán todas las pruebas de aceptación que se necesiten para asegurar su correcto funcionamiento.

3.3.1.2 Iteración 1

Caso de Prueba de aceptación.	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Establecer conexión SSH.	
Descripción: La aplicación debe permitir establecer la conexión con el servidor mediante el protocolo SSH.	
Condiciones de ejecución: Se debe tener instalado OpenSSH en la máquina local y en el servidor.	
Pasos de ejecución: El módulo se conecta al servidor de base de datos mediante SSH introduciendo nombre de usuario, IP del servidor y contraseña.	
Resultados esperados: Se realiza la conexión satisfactoriamente.	
Evaluación de la prueba: Satisfactoria.	

Tabla 29: Prueba de aceptación 1.

Caso de Prueba de aceptación.	
Código: HU2_P1	**Historia de usuario:** 2
Nombre: Listar paquetes existentes en los repositorios.	
Descripción: La aplicación debe mostrar todos los paquetes que se encuentran en los repositorios.	

Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.
Pasos de ejecución: Se intenta listar los paquetes.
Resultados esperados: Se listan los paquetes de los repositorios.
Evaluación de la prueba: Satisfactoria.

Tabla 30: Prueba de aceptación 2.

Caso de Prueba de aceptación.	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Listar paquetes instalados en el servidor.	
Descripción: La aplicación debe mostrar todos los paquetes que se encuentran instalados en el servidor.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta listar los paquetes.	
Resultados esperados: Se listan los paquetes instalados.	
Evaluación de la prueba: Satisfactoria	

Tabla 31: Prueba de aceptación 3.

Caso de Prueba de aceptación.	
Código: HU3_P1	**Historia de usuario:** 3
Nombre: Instalar paquetes.	
Descripción: La aplicación debe permitir instalar los paquetes seleccionados.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta instalar los paquetes seleccionados.	
Resultados esperados: Se instalan los paquetes seleccionados.	
Evaluación de la prueba: Satisfactoria	

Tabla 32: Prueba de aceptación 4.

Caso de Prueba de aceptación.	
Código: HU3_P2	**Historia de usuario:** 3
Nombre: Mostrar información del proceso de instalación.	
Descripción: La aplicación debe mostrar la información relacionada con el proceso de instalación.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta mostrar la información del proceso de instalación.	
Resultados esperados: Se muestra la información.	
Evaluación de la prueba: Satisfactoria	

Tabla 33: Prueba de aceptación 5.

3.3.1.2 Iteración 2

Caso de Prueba de aceptación.	
Código: HU4_P1	**Historia de usuario:** 4
Nombre: Desinstalar paquetes.	
Descripción: La aplicación debe permitir desinstalar los paquetes seleccionados.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta desinstalar los paquetes seleccionados.	
Resultados esperados: Se desinstalan los paquetes seleccionados.	
Evaluación de la prueba: Satisfactoria	

Tabla 34: Prueba de aceptación 6.

Caso de Prueba de aceptación.	
Código: HU4_P2	**Historia de usuario:** 4
Nombre: Mostrar información del proceso de desinstalación.	
Descripción: La aplicación debe mostrar la información relacionada con el proceso de desinstalación.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	

Pasos de ejecución: Se intenta mostrar la información del proceso de desinstalación.
Resultados esperados: Se muestra la información.
Evaluación de la prueba: Satisfactoria

Tabla 35: Prueba de aceptación 7.

Caso de Prueba de aceptación.	
Código: HU5_P1	**Historia de usuario:** 5
Nombre: Actualizar el sistema operativo.	
Descripción: La aplicación debe permitir actualizar el sistema operativo del servidor.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta actualizar el sistema operativo del servidor.	
Resultados esperados: Se actualiza el sistema operativo.	
Evaluación de la prueba: Satisfactoria	

Tabla 36: Prueba de aceptación 8.

Caso de Prueba de aceptación.	
Código: HU5_P2	**Historia de usuario:** 5
Nombre: Mostrar información del proceso de actualización.	
Descripción: La aplicación debe mostrar la información relacionada con el proceso de actualización.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta mostrar la información del proceso de actualización.	
Resultados esperados: Se muestra la información.	
Evaluación de la prueba: Satisfactoria	

Tabla 37: Prueba de aceptación 9.

3.3.1.3 Iteración 3

Caso de Prueba de aceptación.

Código: HU6_P1	**Historia de usuario:** 6
Nombre: Agregar nuevos repositorios.	
Descripción: La aplicación debe permitir agregar nuevos repositorios.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta agregar nuevos repositorios.	
Resultados esperados: Se actualiza el sistema operativo.	
Evaluación de la prueba: Satisfactoria	

Tabla 38: Prueba de aceptación 10.

Caso de Prueba de aceptación.	
Código: HU6_P2	**Historia de usuario:** 6
Nombre: Eliminar repositorios.	
Descripción: La aplicación debe permitir eliminar un repositorio seleccionado.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta eliminar repositorios.	
Resultados esperados: Se elimina el repositorio seleccionado.	
Evaluación de la prueba: Satisfactoria	

Tabla 39: Prueba de aceptación 11.

Caso de Prueba de aceptación.	
Código: HU6_P3	**Historia de usuario:** 6
Nombre: Editar repositorios.	
Descripción: La aplicación debe permitir editar un repositorio seleccionado.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta editar el repositorio seleccionado.	
Resultados esperados: Se edita el repositorio seleccionado.	

Evaluación de la prueba: Satisfactoria

Tabla 40: Prueba de aceptación 12.

Caso de Prueba de aceptación.	
Código: HU7_P1	**Historia de usuario:** 7
Nombre: Mostrar información de los paquetes.	
Descripción: La aplicación debe permitir mostrar la información de los paquetes seleccionados.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta mostrar la información del paquete seleccionado.	
Resultados esperados: Se muestra la información del paquete seleccionado.	
Evaluación de la prueba: Satisfactoria	

Tabla 41: Prueba de aceptación 13.

Caso de Prueba de aceptación.	
Código: HU8_P1	**Historia de usuario:** 8
Nombre: Hacer búsqueda de paquetes.	
Descripción: La aplicación debe permitir buscar un paquete determinado.	
Condiciones de ejecución: El usuario debe haber establecido correctamente la conexión con el servidor.	
Pasos de ejecución: Se intenta buscar un paquete determinado.	
Resultados esperados: Se muestra el paquete.	
Evaluación de la prueba: Satisfactoria	

Tabla 42: Prueba de aceptación 14.

3.4 Conclusiones

En el presente capítulo se abordaron los temas referentes a la implementación de la solución propuesta y las pruebas realizadas a la misma con el objetivo de evaluar la calidad del producto y determinar el nivel de conformidad del cliente. Se realizaron

todas las tareas relacionadas con las historias de usuarios y las pruebas de aceptación obteniendo como resultado una aplicación funcional que satisface la propuesta presentada en el presente trabajo.

Conclusiones.

Una vez concluida la investigación se concluye que:

- Se elaboró el diseño metodológico de la investigación, dando paso al análisis de las principales herramientas utilizadas para la gestión de paquetes.
- Se cumplieron los objetivos trazados obteniendo como resultado el desarrollo de un módulo para SASEP que permite la gestión de paquetes mediante una interfaz visual y de forma remota.
- La elaboración de una herramienta con interfaz visual para la gestión remota de paquetes facilita el trabajo a los gestores de configuración y evita la ocurrencia de errores.
- El uso de la metodología XP favoreció la elaboración de la herramienta.
- Con las pruebas de aceptación se pudo comprobar el grado de satisfacción del cliente y se demostró que el software está lista para su uso.

Bibliografía

1. **Real Academia Española.** Real Academia Española. [En línea] 2001. [Citado el: 15 de 1 de 2012.] http://buscon.rae.es/draeI/.

2. **EMOPA.** EmopaGreen. [En línea] [Citado el: 15 de 1 de 2012.] http://www.emopagreen.com/instalacion.html.

3. **Real Academia Española.** Real Academia Española. [En línea] [Citado el: 15 de 1 de 2012.] http://buscon.rae.es/draeI/.

4. **Geofísica UNAM.** Geofísica UNAM. [En línea] [Citado el: 20 de 1 de 2012.] Artículo consultado en sitio web. http://mmc.geofisica.unam.mx/LuCAS/Manuales-LuCAS/RHAT/rhl-ig-6.0es/node290.html.

5. **Centro Nacional de Electromagnetismo Aplicado.** CNEA Electromagnetismo Aplicado. [En línea] 2011. [Citado el: 15 de 1 de 2012.] Artículo de Sitio Web. http://www.cnea.uo.edu.cu/index.php?option=com_content&view=article&id=127:repositorios-digitales&catid=1:ultimas-noticias&Itemid=156.

6. **Spinellis, Diomidis.** IEEE Computer Society. [En línea] 3 de 2012. [Citado el: 2 de 4 de 2012.] IEEE Software magazine Tools of the Trade column. http://www.computer.org/portal/web/csdl/abs/html/mags/so/2012/02/mso2012020084.htm.

7. **openSUSE.** openSUSE. [En línea] 2011. [Citado el: 2 de 4 de 2012.] http://en.opensuse.org/Package_management.

8. **Proyecto Debian.** *Manual de APT.* 1998. Manual de usuario. Man - Una interfaz de los manuales de referencia electrónicos..

9. **Burrows, Daniel.** *Manual de aptitude.* 2011. Manual de usuario. Man - Una interfaz de los manuales de referencia electrónicos..

10. **Brown, Robert G.** Duke Physics. [En línea] 17 de Diciembre de 2003. [Citado el: 4 de abril de 2012.] http://www.phy.duke.edu/~rgb/General/yum_article/yum_article.pdf.

11. **Synaptic.** Synaptic. [En línea] 7 de 1 de 2009. [Citado el: 20 de 1 de 2012.] Artículo en sitio web.. http://www.nongnu.org/synaptic/.

12. **Higginson, Andrew.** *Manual de Ubuntu Software Center.* Manual de Usuario. Man - Una interfaz de los manuales de referencia electrónicos..

13. **Raleigh, Varsity Drive.** Massachusetts Institute of Technology. [En línea] http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html. [Citado el: 21 de 1 de 2012.]

14. **Foundation, The Eclipse.** The Eclipse Foundation open source community website. [En línea] 2012. [Citado el: 5 de 5 de 2012.] http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/int_eclipse.htm.

15. **Corporation, Nokia.** Qt. [En línea] 2012. [Citado el: 2012 de 5 de 5.] http://qt.nokia.com/products/eclipse-integration.

16. **Team, Code::Blocks.** Code::Blocks. [En línea] 2011. [Citado el: 5 de 5 de 2012.] http://www.codeblocks.org/.

17. **Semiconductors, NXP.** NXP Low Power RF. [En línea] 2012. [Citado el: 5 de 5 de 2012.] http://www.jennic.com/index.php.

18. **KDevelop team.** KDevelop. [En línea] 3 de Jun de 2011. [Citado el: 5 de May de 20.] http://kdevelop.org/frontpage.

19. **Nokia.** Qt. [En línea] 2012. [Citado el: 16 de May de 2012.] http://qt.nokia.com/products/developer-tools.

20. **Pressman, Roger S.** *Software Engineering.* [ed.] McGraw-Hil. Séptima edición. New York : McGraw-Hil, 2010.

21. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000.

22. **Schwaber, Ken.** Scrum.org. [En línea] 2012. http://www.scrum.org/what-is-scrum.

23. **Wells, Don.** Extreme Programming. [En línea] 2009. [Citado el: 5 de Mayo de 2012.] http://www.extremeprogramming.org/.

24. **Joskowicz, Jose.** Instituto de Ingeniería Eléctrica. [En línea] 10 de 2 de 2008. [Citado el: 2 de 4 de 2012.] http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

25. **H. Steinberg, Daniel y Daniel W., Palmer.** *Extreme Software Engineering.*

26. **Solís., Manuel Calero.** *Una explicación de la programación extrema (XP).* Madrid : s.n., 2003.

27. **Joskowicz, José.** *Reglas y Prácticas en eXtreme Programming.* Universidad de Vigo, España. 2008.

28. **Hetzel, William C.** *The Complete Guide to Software Testing.* Wellesley, Massachusetts : QED Information Sciences, 1988. 0894352423.

29. **Pan, Jiantao.** Electrical & Computer Engineering. [En línea] 1999. [Citado el: 5 de abril de 2012.] Universidad Carnegie Mellon. http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/#reference.

30. UTN-FRT. [En línea] [Citado el: 21 de 1 de 2012.] http://www.frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm#_INTRODUCCI%C3%93N.

31. **IEEE.** *Compilation of IEEE Standard Computer Glossaries.* 1991.

32. **Sommerville, Ian.** *Software Engineering.* 2006. 0-321-31379-8.

33. **Quiroga, Juan Pablo.** Universidad de los Andes. [En línea] [Citado el: 4 de 3 de 2012.] http://sistemas.uniandes.edu.co/~csof5101/dokuwiki/lib/exe/fetch.php?media=principal:csof5101-requerimientos.pdf.

34. **Microsoft.** MSDN. [En línea] [Citado el: 21 de 1 de 2012.] http://msdn.microsoft.com/es-es/vbasic/ms789102.

35. **Oracle.** JavaTM. [En línea] [Citado el: 21 de 1 de 2012.] http://www.java.com/es/download/faq/whatis_java.xml.

36. **Schwaber, K.** *Advanced Development Methods. SCRUM Development Process Retrieved.* 2006.

37. **Chin, Gary.** *Agile Project Management: How to Succeed in the Face of Changing Project Requirements.* 2004.

38. **Nokia.** http://doc.qtsoftware.com/qtcreator-1.1/creator-quick-tour.html. [En línea]

39. —. qt.nokia.com/products/developer-tools. [En línea]

40. **Duke University.** Duke Physics. [En línea] [Citado el: 4 de abril de 2012.] http://www.phy.duke.edu/~rgb/General/yum_HOWTO/yum_HOWTO/yum_HOWTO-1.html.

41. **EcuRed.** EcuRed. [En línea] [Citado el: 20 de 1 de 2012.] http://www.ecured.cu/index.php/Repositorio_de_GNU/Linux.

42. —. EcuRed. [En línea] [Citado el: 20 de 1 de 2012.] http://www.ecured.cu/index.php/Ubuntu.

43. **Wikipedia.** Wikipedia. [En línea] [Citado el: 6 de abril de 2012.] http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.

44. **IEEE.** *Compilation of IEEE Standard Computer Glossaries.* 1991.

45. **Joskowicz, Ing. José.** Instituto de Ingeniería Eléctrica. [En línea] 10 de 2 de 2008. [Citado el: 4 de 2 de 2012.] http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

46. **ter Hofstede, Arthur y van der Aalst, Wil.** Workflow Patterns. [En línea] 2011. [Citado el: 3 de 4 de 2012.] http://www.workflowpatterns.com/.