Universidad de las Ciencias Informáticas

Facultad 1

***Título**: Identificación de requisitos a partir de un repositorio de aplicaciones.*

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

***Autor**: Manuel Enrique Peiso Cruz*

***Tutores**:*

Mtr. Yusleydi Fernández del Monte.

Ing. Michel Evaristo Febles Parker.

La Habana

Junio 2015

Declaración de Autoría

Declaro ser el único autor del presente trabajo y autorizo a la Facultad 1 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Manuel Enrique Peiso Cruz

_____

Firma del autor(a)

Mtr. Yusleydi Fernández del Monte

_____

Firma del Tutor(a)

Ing. Michel Evaristo Febles Parker

_____

Firma de Tutor(a)

Agradecimientos

A mi mamá, por haberme brindado una vida tan feliz a pesar de las adversidades y por ser la madre más simpáticamente genial que se pueda tener.

A mi papá, por constituir mi meta a seguir profesionalmente, por enseñarme la importancia del estudio.

A mi abuela "Manicita", por darme tanto amor y cariño.

A todo el familión que me rodeó en mi infancia, hermano, primos, familia en general, por no dejarme tranquilo ni un segundo y por soportar mi hiperactividad.

A súper Leo, por todo el apoyo, el cariño y la comprensión que ha depositado en mí.

A Neyvis, quien me ha sacado del paso tantas veces pero como la quiero tanto no puedo pelearme con ella. Gracias por estar conmigo en todo momento. A Arian, el primer amigo que tuve en esta universidad y que hoy sigue siendo mi gran compañero. Gracias por no darme la espalda cuando me equivoqué. A Danilo, porque sin sus locuras, ocurrencias y su forma de ser en general esta universidad no hubiera sido lo mismo. Gracias por hacerme reír hasta cuando estaba triste. A Yiso, aunque fue una de las últimas en incorporarse a la FAMILIA ha sabido ganarse un pedacito de todos nuestros corazones. A Daniel, por tener un cariño tan grande que compartir con nosotros. A Yennis y Lisdy, por compartir conmigo tantos momentos y por su simpatía. A Roger, por estar siempre listo a decir un disparate inapropiado y por sacarnos del paso cada vez que tuviste oportunidad, sin eso nada hubiera sido lo mismo. Al resto de mis

amigos del grupo, porque todos han compartido conmigo muchos momentos felices y otros no tanto en estos cinco años, porque hemos sido un grupo humilde, respetuoso y en el cual hemos llegado a conocer y comprender cada una de las diferencias que nos caracterizan.

Resumen

La identificación de requisitos es una tarea de vital importancia que permite determinar cuáles son las funcionalidades que una aplicación debe poseer y bajo qué condiciones se debe instaurar. En el Centro de *Software* Libre de la Universidad de las Ciencias Informáticas se desarrolla la distribución GNU/Linux Nova que, como todas las distribuciones de este tipo, almacena las aplicaciones en un repositorio de código fuente y binario en forma de paquetes, pero existen problemas a la hora de realizar las actualizaciones de dichas aplicaciones, pues muchas veces no se puede determinar cuáles requisitos quedaron sin implementar y hay que hacer, para su identificación, un análisis desde cero. La presente investigación propone el desarrollo de un complemento para la herramienta "Metodología para el desarrollo de distribuciones GNU/Linux" que permita automatizar el proceso de identificación de requisitos a la hora de actualizar las aplicaciones que se encuentran alojadas en el repositorio. Con ese propósito, se estudia la manera en que varias metodologías de desarrollo de *software* tratan la identificación de requisitos, así como la estructura y actividades dentro de un repositorio para identificar elementos útiles a tener en cuenta. Para la implementación de la propuesta de solución se utiliza el sistema de gestión de contenidos Drupal, PostgreSQL como sistema gestor de base de datos y Apache como servidor web. Finalmente, se obtuvo como resultado práctico un complemento que identifica los requisitos que le faltan por implementar a las aplicaciones que se encuentran alojadas en un repositorio, a través de la incorporación de un archivo en los paquetes que el mismo alberga.

Palabras clave: identificación de requisitos, repositorios de códigos fuente y binarios.

Índice

Introducción

El *software* libre ha ganado terreno en la esfera informática debido a las facilidades que brinda: distribuir, modificar, estudiar y mejorar las aplicaciones que utiliza (1), por ello cada vez son más los que se interesan por este tipo de tecnología, para de esta forma disminuir las dependencias tecnológicas que afectan fundamentalmente a los países del tercer mundo.

Las distribuciones de GNU/Linux tienen la delantera, si de temas de *software* libre se trata. Estas brindan muchas facilidades: costos menores de inversión, más seguridad y menos limitaciones en la automatización de sus procesos (2). Además utilizan repositorios de códigos fuente y binarios, donde se aloja información que representa tanto el código fuente de las aplicaciones, como los binarios que se utilizan para ser instalados en el Sistema Operativo (SO) del usuario. Las distribuciones más reconocidas a nivel internacional cuentan con grandes repositorios de códigos fuente y binarios que se actualizan periódicamente y que son mantenidos por equipos especializados y con una alta preparación debido a la importancia que poseen.

En Cuba, la migración hacia estándares de *software* libre y código abierto es una de las tareas fundamentales que se realiza con el objetivo de eliminar las dependencias que el *software* privativo genera con las transnacionales informáticas. Vale aclarar que el *software* privativo se compra a precios elevados y requiere de la adquisición de licencias periódicamente o perpetuas que suelen ser extremadamente costosas.

Debido a esto y a las facilidades que las distribuciones GNU/Linux poseen, en el año 2007 se lanza la versión 1.0 de GNU/Linux Nova (2), que ha avanzado hasta su versión 4.0, una distribución completamente nacional que fue desarrollada en el Centro de *Software* Libre (CESOL) de la Universidad de las Ciencias Informáticas (UCI).

Esta distribución cubana, desde su creación, sirve de apoyo para el proceso de migración del país, por lo que cuenta con un equipo de especialistas multidisciplinario y con una elevada

preparación que se encarga del constante desarrollo y perfeccionamiento del sistema. Cuenta también con un repositorio donde se albergan las aplicaciones que necesita el sistema para cumplir con las necesidades del usuario nacional; en tanto, el equipo de especialistas del Departamento de Sistemas Operativos (DSO) de CESOL es el encargado de su desarrollo y mantenimiento. Para ellos es fundamental un control minucioso de los requisitos funcionales y no funcionales implementados y los que faltan por implementar de las aplicaciones informáticas que se encuentran alojadas en su repositorio, ya que con esto se logra que las actualizaciones a desarrollar tengan la planificación, calidad y control necesarios.

La importancia de los requisitos recae en que los mismos son directamente proporcionales a las funcionalidades que van a ser implementadas en cualquier sistema y a las restricciones de *software*, *hardware* y seguridad que van a mejorar su funcionamiento. Estas funcionalidades se identifican a través de la comprensión del proceso que se quiera informatizar y de lo que el cliente espera del sistema. En ocasiones, no se implementan todos los requisitos que se prevén para una aplicación informática, sino que se priorizan algunos de mayor importancia para obtener, en un menor tiempo, un prototipo funcional que pueda satisfacer parcialmente las necesidades del cliente y se dejan otros, de menor importancia, para futuras versiones o actualizaciones del producto. De igual manera, durante el proceso de desarrollo de una aplicación informática es común que surjan nuevas funcionalidades que, de incorporarse al sistema, pueden mejorar su rendimiento y eficiencia, de las cuales también algunas se pueden dejar para otras versiones. Como se puede apreciar, la cantidad de requisitos aumenta a lo largo de todo el proceso de desarrollo de *software*, por lo que es necesario un control detallado de los mismos para conocer cuánto se ha hecho y cuánto falta por hacer. Si a esto se suma que el crecimiento de las aplicaciones informáticas que se albergan en los repositorios es cada vez mayor, se torna aún más complicado determinar hasta qué punto están desarrolladas las mismas. Al proceso que ayuda a los ingenieros de *software* a identificar, comprender y controlar los

requisitos de un sistema, se le denomina ingeniería de requisitos (3). En este sentido existen herramientas que realizan este trabajo (Visual Paradigm, OSRMT, Rational RequisitePro) y que son muy usadas en las grandes compañías informáticas del mundo.

En CESOL, para la actualización de las aplicaciones informáticas que desarrollan, se utiliza la opinión de los especialistas con mayor experiencia en el proyecto para determinar los requisitos que faltan por implementar, lo cual constituye una práctica ineficiente si se tiene en cuenta que algunos no dominan los temas relacionados con la aplicación por no haber estado vinculados con su desarrollo y otros, aunque estuvieron vinculados, no recuerdan con exactitud qué fue lo que quedó sin implementar. También se estudia documentación almacenada en el expediente del proyecto que brinda información referente a los requisitos implementados, para saber hasta qué punto están desarrollados, pero existe información insuficiente sobre los que dejaron de implementarse.

En la investigación se determinó que el mecanismo utilizado en CESOL para obtener un listado de los requisitos que faltan por desarrollar en aplicaciones informáticas del repositorio que utilizan es ineficiente, lo cual provoca el aumento del tiempo de desarrollo de las actualizaciones, al tener que realizar, en la mayoría de los casos, la identificación de los mismos desde cero. Muchas veces quedan requisitos que no se tienen en cuenta por la falta de control que existe en el proceso de desarrollo de *software*, lo que provoca inconformidades del cliente, que en ocasiones no ve satisfechas sus expectativas.

Ante la situación planteada se ha establecido el siguiente **problema de investigación**: ¿Cómo identificar los requisitos que faltan por implementar en aplicaciones informáticas que se encuentran alojadas en un repositorio de código fuente y binario? A partir del problema de investigación planteado, se define como **objeto de estudio** el proceso de identificación de requisitos. Para desarrollar la investigación, se establece como **objetivo general:** desarrollar una aplicación informática para la identificación de requisitos que faltan por implementar en

aplicaciones que se encuentran alojadas en un repositorio de código fuente y binario. Como **campo de acción** se define el proceso de identificación de requisitos en la actualización de las aplicaciones que forman parte de un repositorio de código fuente y binario.

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

- Sistematizar el estudio sobre la ingeniería de requisitos en el desarrollo de *software* teniendo en cuenta el trabajo con repositorios de códigos fuente y binarios.
- Desarrollar una aplicación informática que permita la identificación de los requisitos que faltan por implementar en aplicaciones que se encuentran alojadas en un repositorio de código fuente y binario.
- Evaluar el funcionamiento de la aplicación con el fin de detectar no conformidades.

La investigación se sustenta en la siguiente **idea a defender**: con el desarrollo de una aplicación informática que permita identificar los requisitos que faltan por implementar en aplicaciones que se encuentran alojadas en un repositorio de código fuente y binario se puede disminuir el tiempo de actualización de las mismas y tener mayor control de dicho proceso.

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

- Revisión de bibliografía asociada a la ingeniería de requisitos y al trabajo con los repositorios de códigos fuente y binarios, para comprender cuál es la relación entre ellos y de qué forma se lleva el control de los requisitos implementados y los que faltan por implementar.
- Definición de los requisitos funcionales y no funcionales que posibilitan entender cuáles son las funcionalidades que el sistema debe poseer y bajo qué condiciones informáticas se debe instaurar.
- Representación de la arquitectura de la herramienta a desarrollar para comprender la manera en que se va a estructurar, de forma tal que pueda integrarse a la herramienta

"Metodología para el desarrollo de distribuciones GNU/Linux".

- Codificación de las funcionalidades definidas para obtener el prototipo funcional de la aplicación informática.
- Diseño y ejecución de casos de prueba que permitan corroborar la importancia de la aplicación.

Para desarrollar esta investigación de manera eficiente se utilizan varios **métodos científicos de investigación**. Estos métodos se dividen en métodos teóricos y métodos empíricos. Entre los **métodos teóricos** se encuentran:

- La **revisión bibliográfica** que permite obtener información sobre los procesos de obtención de requisitos, el funcionamiento y estructura interna de los repositorios, así como la metodología Nova - OpenUp y la herramienta "Metodología para el desarrollo de distribuciones GNU/Linux".
- El **Analítico–Sintético** con el fin de descomponer el problema de investigación en elementos por separado (identificación de requisitos y repositorios de códigos fuente y binarios) y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.
- El **Inductivo-Deductivo** es utilizado con el fin de, a partir del estudio de hechos aislados referidos a la manera en que se realiza la identificación de requisitos, arribar a proposiciones generales para inferir cómo se debe realizar en la propuesta de solución, a través de un razonamiento lógico.

Y entre los **métodos empíricos** se encuentra:

- La **Entrevista** para valorar la situación existente en la detección de requisitos desde un repositorio de código fuente y binario en CESOL, lo que ayuda a determinar sus ineficiencias y la manera en que puede cambiar al introducir la solución propuesta.
- El **Modelado** que permite la representación de la estructura de la aplicación a desarrollar.

Este documento se encuentra organizado en tres capítulos, a continuación se muestra una breve descripción de los mismos:

En el **primer capítulo** titulado **La identificación de requisitos y su vinculación con los repositorios de códigos fuente y binarios**, se definen los principales conceptos asociados al dominio del problema que son indispensables para el desarrollo y comprensión de esta investigación. Contiene además un análisis de herramientas y tecnologías para definir las que se van a utilizar en el desarrollo de la aplicación.

En el **segundo capítulo** titulado **Requisitos y arquitectura de la aplicación para la identificación de requisitos en un repositorio de código fuente y binario**, se realiza todo lo vinculado con la identificación de los requisitos y la arquitectura del sistema, describiendo de manera más detallada los elementos que se deben obtener en cada una de las fases propuestas por la metodología de desarrollo de *software* a utilizar.

En el **tercer capítulo** titulado **Implementación y evaluación de la aplicación para la identificación de requisitos en un repositorio de código fuente y binario**, se explican temas referentes a la implementación de la solución. Además se realizan los casos de pruebas para lograr el desarrollo de un *software* con la calidad requerida.

Capítulo 1: La identificación de requisitos y su vinculación con los repositorios de códigos fuente y binarios.

En el presente capítulo se abordan los conceptos fundamentales que sustentan la presente investigación, se analiza la manera en que se realiza la identificación de requisitos en el DSO para determinar vulnerabilidades y necesidades y se realiza un resumen de las principales metodologías de desarrollo de *software,* para conocer la manera en que tratan la identificación de requisitos con el objetivo de determinar los aspectos a tener en cuenta en la propuesta de solución. También se estudia la estructura, actividades e información que se alberga en los repositorios de códigos fuente y binarios para determinar la manera en que se pueden identificar los requisitos que le faltan por implementar a sus aplicaciones. Se definen las herramientas, lenguajes de programación, tecnologías y metodología de desarrollo de *software* a utilizar.

1.1 Definiciones de interés

1.1.1 Aplicación web

Pressman expone que las aplicaciones web son sistemas informáticos que están publicados en la *world wide web* y que permiten ofrecer capacidades de cálculos junto con información. Son sofisticadas herramientas de computación que no solo proporcionan función por sí mismas al usuario final, sino que también se han integrado con bases de datos corporativas y aplicaciones de negocio (4).

Según Isabel Sánchez, las aplicaciones web son programas que permiten la interacción dinámica entre los usuarios y la información o servicios. Estas pueden ser, desde aplicaciones encargadas de gestionar la información hasta aplicaciones complejas, capaces de realizar las mismas funciones que los programas para ordenador (1).

En la web se puede encontrar que una aplicación web es un conjunto de páginas que interactúan unas con otras y con diversos recursos en un servidor web, incluidas bases de datos. Esta interacción permite implementar características en su sitio como catálogos de productos virtuales, administradores de noticias y contenidos. Adicionalmente se pueden realizar consultas a bases de datos, registrar e ingresar información, solicitudes, pedidos y múltiples tipos de información en línea en tiempo real (5).

Se puede concretar que una aplicación web es un conjunto de páginas que interactúan entre sí y con otros recursos informáticos, como bases de datos, para gestionar información o para ejecutar tareas complejas similares a las de los programas para ordenador.

1.1.2 Complemento o *plugin*

Diferentes fuentes expresan que un complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la interfaz visual (6).

Otras fuentes por su parte lo definen como un programa que puede anexarse a otro para aumentar sus funcionalidades (generalmente sin afectar otras funciones ni afectar la aplicación principal). No se trata de un parche ni de una actualización, es un módulo aparte que se incluye opcionalmente en una aplicación (7).

De donde *plugin* o complemento es un programa informático que se puede incorporar a otro con el objetivo de aumentar las funcionalidades que este último brinda.

1.1.3 Módulo

Según lo planteado por Edgar Pedro García Achillo, en programación un módulo es una porción de un programa de computadora. De las varias tareas que debe realizar un programa para cumplir con su función u objetivos, un módulo realiza, comúnmente, una de dichas tareas (o

varias, en algunos casos) (8).

En la web se puede encontrar que en programación, un módulo es un *software* que agrupa un conjunto de subprogramas y estructuras de datos. Los módulos son unidades que pueden ser compiladas por separado, lo cual los hace reusables y permite que múltiples programadores trabajen en diferentes módulos en forma simultánea, produciendo ahorro en los tiempos de desarrollo (9).

Por lo que se puede definir que módulo es un programa informático que se integra, y por tanto forma parte, de un sistema mayor. El mismo se encarga de realizar tareas específicas que se encuentran dentro de tareas más generales implementadas por el sistema padre.

Se decide realizar un complemento, pues el sistema a desarrollar persigue aumentar las funcionalidades que ofrece una aplicación ya existente. Este sistema debe incorporarse a la aplicación antes mencionada de forma opcional y ser deshabilitado en caso de ser necesario.

1.2 Tecnologías, herramientas, metodología y lenguajes a utilizar

1.2.1 Sistema de gestión de contenidos

Con esta denominación se conoce una herramienta de *software* que permite crear, organizar y publicar documentos y otros contenidos de forma colaborativa. Los sistemas de gestión de contenidos (CMS por sus siglas en inglés, c*ontent management system*) están formados por un conjunto de aplicaciones web que, de un modo similar a un portal, operan tanto en internet como en una intranet. Su principal ventaja consiste en el hecho de que permiten organizar y mostrar contenidos sin que sea necesario poseer grandes conocimientos de programación web (10).

1.2.2 Representación del sistema de gestión de contenidos a utilizar

Drupal:

Es un CMS libre, modular, multipropósito y muy configurable que permite publicar artículos, imágenes, archivos y otros servicios añadidos como foros, encuestas, votaciones, blogs y administración de usuarios y permisos. Drupal es un sistema dinámico, en lugar de almacenar sus contenidos en archivos estáticos en el sistema de ficheros del servidor de forma fija, el contenido textual de las páginas y otras configuraciones son almacenados en una base de datos y se editan utilizando un entorno web (11).

Es un programa con licencia GNU/GPL[1], escrito en PHP, combinable con MySQL y PostgreSQL, además de ser desarrollado y mantenido por una activa comunidad de usuarios. Se destaca por la calidad de su código y de las páginas generadas, el respeto de los estándares de la web, y un énfasis especial en la usabilidad y consistencia de todo el sistema (11).

Principales características:

- Ayuda en línea: un robusto sistema de ayuda en línea.
- Búsqueda: todo el contenido es totalmente indexado en tiempo real y se puede consultar en cualquier momento.
- Código abierto: disponible bajo los términos de la licencia GNU/GPL.
- Módulos: la comunidad de Drupal ha contribuido con muchos módulos que proporcionan disímiles funcionalidades.
- Personalización: un robusto entorno de personalización está implementado en su núcleo (12).

Como se puede apreciar, es un CMS de código abierto, ampliamente utilizado en el mundo, por lo que cuenta con una amplia comunidad de usuarios que se encargan de su mantenimiento y constante desarrollo que se encuentra presente incluso en la universidad, haciendo más alcanzable la información y el intercambio con especialistas. Si se tiene en cuenta que el complemento a desarrollar debe ser integrado a una aplicación desarrollada en Drupal, su

1 GNU/GPL: es una licencia que da la libertad de usar, estudiar, compartir (copiar) y modificar el *software*.

utilización va a contribuir a mantener una homogeneidad tanto en la interfaz visual como en la estructura de la base de datos. Se utiliza la versión 7.26.

1.2.3 Lenguajes de desarrollo

HTML (*hypertext markup language* o lenguaje para marcado de hipertexto) es el lenguaje que permite describir y dar forma a las páginas web, además de publicar páginas con fotos, listas, tablas, la obtención de información de los visitantes del sitio, diseñar los formularios para contactar futuros clientes, crear un sitio para vender productos o servicios, incluir videos clip, música, sonidos, y otras aplicaciones que dan vida a las páginas web (13). Se utiliza, en su versión 5, para la construcción de las vistas que forman parte del complemento a desarrollar.

PHP (*hypertext pre-processor***)** es un lenguaje de programación de uso general de código del lado del servidor, originalmente diseñado para el desarrollo web de contenido dinámico. Se puede incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. Se conecta a servidores de bases de datos tales como MySQL, Oracle, Informix y PosgreSQL (14). Se utiliza, en su versión 5, para la implementación de las funcionalidades básicas del sistema.

CSS (hojas de estilo en cascada) es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos (15). Se utiliza, en su versión 3, para darle formato a las páginas web de forma tal que el diseño de las mismas sea homogéneo y agradable.

Java Script es un lenguaje ligero e interpretado, orientado a objetos con funciones de primera clase, más conocido como el lenguaje de script[2] para páginas web**.** Es un lenguaje

2 script: es un programa que por lo regular se almacena en un archivo de texto plano.

multiparadigma[3], basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa (16). Se usa, en su versión 5, para crear efectos atractivos y dinámicos en las páginas web que forman parte del complemento, así como para validar los datos insertados en los formularios.

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional, es multiplataforma (17). Se utiliza, en su versión 2.7.6, para la creación de scripts que trabajan directamente con el repositorio y que son usados por el código php para realizar las funcionalidades.

1.2.4 Servidores web

Es un programa que se ejecuta continuamente en un computador, manteniéndose a la espera de peticiones de ejecución que le hace un cliente o un usuario de la red. El servidor web se encarga de contestar a estas peticiones de forma adecuada, entregando como resultado una página web o información de todo tipo de acuerdo a los comandos solicitados. En este punto es necesario aclarar lo siguiente: mientras que comúnmente se utiliza la palabra servidor para referirse a una computadora con un *software* servidor instalado, en estricto rigor un servidor es el *software* que permite la realización de las funciones descritas (18).

Apache es un servidor de red para el protocolo HTTP, elegido para poder funcionar como un proceso independiente, sin que eso solicite el apoyo de otras aplicaciones o directamente del usuario. Apache se distribuye como *software* libre de código abierto, modular, multiplataforma, extensible, popular (fácil de conseguir ayuda/soporte) y gratuito. Permite su uso comercial y no comercial (19).

3 multiparadigma: que soporta más de un paradigma de programación (orientado a objetos, funcional, lógico, declarativo)

Características:

- Altamente configurable, de diseño modular y permite la creación y gestión de registros (*logs*).
- Personaliza la respuesta ante los posibles errores que puedan ocurrir.

Se selecciona el servidor web Apache, debido a que es el utilizado en la aplicación a la que se debe integrar la presente propuesta de solución, además de la gran gama de característica que evidencia. Se utiliza la versión 2.2.

1.2.5 Sistema de gestión de bases de datos

PostgreSQL es un sistema de gestión de bases de datos (SGBD) objeto-relacionales que sigue actualmente un activo proceso de desarrollo a nivel mundial, gracias a un equipo de desarrolladores y contribuidores de código abierto. Dicho sistema es considerado como una de las alternativas de sistema de bases de datos libres, siendo ampliamente popular e ideal para tecnologías web, además de ser fácil de administrar. También ofrece una sintaxis SQL estándar y sencilla, es multiplataforma. Igualmente posee una interfaz para el trabajo con varios lenguajes de programación como: C, C++, Java, *Delphi*, *Python*, Perl, PHP y Bash.
PostgreSQL es un gestor robusto y por ende hoy en día es muy usado con respecto a gestores libres existentes como SQLite, MySQL y FireBird (21).
Es seleccionado debido a que fue el SGBD utilizado en la aplicación a la cual se va a integrar el presente complemento, por lo que su utilización contribuye a la homogeneidad de la información, además de las facilidades antes mencionadas. Se utiliza la versión 9.1.

1.2.6 Herramientas a utilizar

PgAdminIII es una aplicación gráfica para el uso del gestor de bases de datos PostgreSQL,

siendo la más completa y popular con licencia *Open Source*[4]. Es capaz de gestionar bases de datos en las versiones más recientes de PostgreSQL, así como otras versiones comerciales. Está diseñada para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples, hasta desarrollar bases de datos complejas. La aplicación también incluye un editor con resaltado de sintaxis, un editor de código de la parte del servidor y un agente para lanzar scripts programados. La conexión al servidor puede hacerse mediante conexión TCP/IP[5], y puede encriptarse mediante SSL (acrónimo de *Secure Sockets Layer-Protocol* de Capa de Conexión Segura) para mayor seguridad. Tiene características interesantes como son: construcción gráfica de consultas, inclusión del *framework* pgScript para el desarrollo de scripts para ejecutar las consultas, buscador de objetos, opciones para habilitar o deshabilitar reglas y borrar o reasignar roles a determinadas bases de datos (23). Se utiliza la versión 1.14.0.

Sublime Text es un editor de código multiplataforma con una interfaz limpia e intuitiva que soporta un gran número de lenguajes (C, C++, C#, CSS, D, Erlang, HTML, Groovy, Haskell, Java, JavaScript, LaTeX, Lisp, Lua, Markdown, Matlab, OCaml, Perl, PHP, *Python*, R, Ruby, SQL, TCL, Textile y XML). Permite tener varios documentos abiertos mediante pestañas, e incluso emplear varios paneles para aquellos que utilicen más de un monitor. También dispone de modo de pantalla completa, para aprovechar al máximo el espacio visual disponible de la pantalla (24). Se utiliza, en su versión 3.0, para la programación de las funcionalidades en los distintos lenguajes de programación y para la confección de las vistas que presentan la información del sistema.

Visual Paradigm (VP) es una herramienta UML (lenguaje de modelado unificado) que está diseñada para una amplia gama de usuarios, incluidos los ingenieros de *software*, analistas de sistemas, analistas de negocios y arquitectos de sistemas, o para cualquier persona que esté

4 Licencia Open Source: permite estudiar, modificar y mejorar el diseño de un *software* mediante la disponibilidad de su código fuente.

5 TCP/IP: sistema de protocolos que hacen posible la transferencia de datos entre dos estaciones.

interesada en la construcción de sistemas de *software* fiables a gran escala con un enfoque orientado a objetos. Además, VP-UML soporta los últimos estándares de la notación UML. Permite producir informes en formato PDF, *word* o en formato HTML sin ningún esfuerzo (25). Se decide usar VP, en su versión 8.0, para modelar los diagramas que se generen en cada una de las etapas del proceso de desarrollo de *software* por ser la herramienta más prestigiosa y eficiente de su tipo en la actualidad, cuenta con una amplia gama de funcionalidades y con una gran facilidad de uso, además de ser la que está definida por la universidad y por la que la misma paga una licencia con fines educacionales.

1.2.7 Metodología de desarrollo de *software*

Las metodologías de desarrollo de *software* son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de *software*.

Van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando qué personas deben participar en el desarrollo de las actividades y qué papel deben tener. Además detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla (26).

SXP

Es un híbrido cubano entre las metodologías ágiles XP y Scrum, que ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de *software* para el mejoramiento de la actividad productiva, fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo y ayudando al líder del proyecto a tener un mejor control del mismo. Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (27).

Consta de 4 fases principales:

- Planificación-definición: donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto.
- Desarrollo: donde se realiza la implementación del sistema hasta que esté listo para ser entregado.
- Entrega: puesta en marcha.
- Mantenimiento: donde se realiza el soporte para el cliente (27).

Se selecciona SXP pues el proyecto a desarrollar es de corta duración y cuenta con un equipo pequeño que se encuentra en continua comunicación con el cliente. Dada las condiciones y facilidades que brinda se determina que sus características se asocian más al tipo de proyecto que se lleva a cabo durante la construcción de una aplicación web. Además, la aplicación a la que se debe integrar el presente sistema utilizó dicha metodología de desarrollo de *software*, así que su uso es adecuado para que la documentación y el proceso ingenieril en general mantengan la misma estructura.

1.3 ¿Cómo se realiza la identificación de requisitos en el DSO?

Según Pressman la identificación de requisitos es una etapa de la ingeniería de requisitos donde se determinan cuáles son los objetivos para el sistema o producto, qué es lo que se debe lograr, de qué forma el producto va a satisfacer las necesidades del negocio y por último cómo se va a utilizar el sistema día a día (3).

Por su parte, Sommerville plantea que la identificación de requisitos es una actividad donde los ingenieros de *software* trabajan con el cliente y usuarios finales para identificar el dominio de la aplicación, qué servicios el sistema debe proveer, las condiciones en las que se debe ejecutar y las restricciones de *hardware* (28).

Por tanto, la identificación de requisitos es una fase de la ingeniería de requisitos en la que el ingeniero de *software*, a través de su interacción con el cliente y de la propia comprensión del

problema, determina cuáles son los requisitos funcionales y no funcionales que el sistema a desarrollar debe poseer para satisfacer las necesidades de los usuarios finales.

Los requisitos de un sistema, según Sommerville, son las descripciones de los servicios que el mismo provee y sus restricciones operacionales. Estos requerimientos reflejan las necesidades del cliente para que el sistema lo ayude a resolver sus problemas. Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de este (28).

En presentación realizada por Saraiva en la universidad de San Judas Tadeo expone que los requisitos de un sistema describen los servicios ofrecidos por un *software* (requisitos funcionales) y sus restricciones de *hardware* (requisitos no funcionales) (29).

Se puede concluir que los requisitos de *software* constituyen las funcionalidades que el sistema debe poseer para satisfacer las necesidades del cliente (requisitos funcionales) así como las restricciones operacionales y de desarrollo que permiten su buen funcionamiento (requisitos no funcionales).

Según entrevista al ingeniero en ciencias informáticas Héctor Pérez Baranda, mantenedor de paquetes, miembro del DSO y especialista con una elevada preparación en los temas relacionados con los repositorios de códigos fuente y binarios, para la identificación de requisitos se realiza una reunión en la que se aplica la técnica tormenta de ideas, para al final, determinar las mejores ideas que puedan aportar al mejoramiento de la aplicación a desarrollar. De igual forma se realiza un estudio de las tendencias globales para incluir nuevas técnicas y métodos que permitan que la aplicación cuente con aspectos de última generación. El Departamento de Servicios Integrales de Migración y Soporte (SIMAYS), a través de la experiencia acumulada en el proceso de migración del país, detecta inconformidades y problemas que necesitan ser mejorados. Esta información también es puesta en manos de los miembros del DSO para que las incluya dentro de los requisitos. Por otra parte, si se trata de la actualización de una aplicación ya

desarrollada en CESOL, se consulta a los especialistas con mayor experiencia en el proyecto y se estudia documentación almacenada en el expediente del proyecto que brinda información referente a los requisitos implementados, para saber hasta qué punto están desarrolladas.

Estos requisitos son procesados utilizando la herramienta VP, aunque existen otras herramientas que son usadas en otras esferas de la universidad con el mismo propósito, como OSRMT (Herramienta de código abierto para la gestión de requisitos) y RequisitePro. A continuación se analizan cada una de ellas para determinar de qué manera realizan la documentación.

1.3.1 Herramienta de código abierto para la gestión de requisitos

Es una herramienta libre para gestionar requisitos que permite el trabajo en equipo, donde cada integrante del grupo de desarrollo de proyectos puede trabajar de forma colaborativa en la misma. También posee la generación de informes a PDF o HTML, según como quiera el usuario.

Cada requerimiento posee diferentes atributos, estos son: requerimiento, número, versión, prioridad, estado, asignación, categoría, complejidad, esfuerzo (horas que incurre en implementarlo), adjuntos (como archivos), descripción, objetivo, contexto, precondición, postcondición, flujos (principal, alterno).

Características de la herramienta:

- Utiliza los casos de uso para relacionarlos con los requerimientos, así facilita la construcción del diagrama de trazabilidad.
- Visualización de la trazabilidad a través de una matriz de relaciones entre las diferentes fases y los requerimientos.
- Espacio para la creación de nuevos artefactos para manejar de una forma personalizada el proyecto y no limitar al usuario a una sola plantilla para cualquier tipo de proyecto.
- Posee una interfaz intuitiva para el usuario (30).

Como se puede apreciar la herramienta es capaz de manejar una serie de informaciones

importantes para un requisito, entre las que sobresale la prioridad, complejidad, esfuerzo, descripción, precondición y objetivo.

1.3.2 Visual Paradigm

Soporta notación UML, lenguaje de modelado de sistemas (SysML) y modelo y notación de procesos de negocio (BPMN). SysML permite la creación de diagramas de requisitos, de igual manera que UML permite gestionar requisitos a través de los diagramas de casos de uso y BPMN a través de los diagramas de actividades. Estos últimos se complementan, pues es posible documentar los casos de uso, utilizando diagramas de actividades, basándose siempre en los modelos del negocio (31).

Por tanto, se puede decir que la herramienta cuenta con una funcionalidad para la especificación de requisitos de *software* que es de gran ayuda para mantener una lista acerca de los requisitos funcionales y no funcionales de un proyecto. Almacena de cada requisito: nombre, descripción, id, tipo (funcional, no funcional), riesgo, estatus, así como las relaciones entre los requisitos. Permite realizar la matriz y el diagrama de trazabilidad (32). Se puede concretar que esta herramienta también almacena información relevante y hace especial énfasis en las relaciones entre requisitos.

1.3.3 Rational RequisitePro

Es una herramienta de administración de requerimientos que puede ser utilizada para administrar requisitos de *software*, la cual promueve la comunicación y colaboración entre los miembros del equipo de trabajo con el fin de reducir los riesgos del proyecto. Esta herramienta permite organizar y priorizar los requerimientos, así como hacer la trazabilidad entre estos.

Para crear los requerimientos, la herramienta solicita los siguientes datos en la pestaña de características generales: tipo (funcional, no funcional), nombre, texto (descripción), paquete

(archivos que componen la funcionalidad), ubicación (en donde va a estar el requerimiento), priorización (alta, media o baja), estado (propuesto, aprobado, incorporado o validado), dificultad (alta, media o baja), id, autor del requerimiento, fecha (de creación), trazabilidad hacia atrás y trazabilidad hacia adelante.

Permite realizar matriz de trazabilidad, árbol de trazabilidad, pero no el grafo de trazabilidad, por lo que no permite detectar caminos críticos (30).

Luego de analizada la manera en que se realiza la identificación de requisitos de las aplicaciones en el DSO, se puede determinar que en el caso de las actualizaciones se realiza la mayor parte del proceso de forma manual y se apoya en valoraciones realizadas por otros trabajadores del departamento que estuvieron vinculados con su desarrollo, pero no existe un proceso automatizado y rápido de realizarlo. De igual forma se estudiaron algunas herramientas de gestión de requisitos, las cuales almacenan de cada requerimiento una serie de información que puede ser tomada en cuenta para incorporarla en la propuesta de solución, todas definen flujos de trabajos a partir de que los requisitos de *software* ya están identificados y ninguna está destinada a automatizar el proceso de identificación de requisitos de las aplicaciones que se encuentran alojadas en un repositorio. Como se puede apreciar, no existe un mecanismo eficiente o herramienta que permita solucionar este problema.

1.4 Identificación de requisitos. CMMI y metodologías de desarrollo de *software*

Todas las metodologías no tratan los requisitos de la misma manera, cada una tiene sus particularidades y propone almacenar distinta información de los mismos en dependencia de la exhaustividad que su proceso de desarrollo propone. De igual forma, la integración de modelos de madurez y capacidades (CMMI), sirve como guía para los ingenieros de *software,* estableciendo pautas que se deben cumplir para que el proceso ingenieril tenga la calidad

necesaria. En este punto, se realiza un análisis acerca de la manera en que algunas metodologías de desarrollo de *software* tratan los requisitos, para identificar cuáles características se podrían tener en cuenta en la aplicación a desarrollar y qué características deben poseer los mismos para cumplir con las políticas propuestas por CMMI.

1.4.1 La integración de modelos de madurez de capacidades

Conjunto de modelos para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de *software*.

Propone dos tipos de requisitos: requisitos del cliente y requisitos del producto (33).

Aspectos a tener en cuenta a la hora de identificar requisitos del cliente:

- Proveedor: quien proporciona el requisito.
- Identificador: elemento identificativo del requisito.
- Modificable: si se puede modificar o no.
- Dependencias: requisitos a tener en cuenta para poder implementar la presente funcionalidad.

Aspectos a tener en cuenta a la hora de identificar requisitos del producto:

- Elementos de entrada.
- Elementos de salida.
- Creador.

Políticas en la obtención de requisitos:

- Obtener el entendimiento de los requisitos: se debe obtener y desarrollar con los clientes el entendimiento del significado de los requisitos.
- Obtener el compromiso a los requisitos: en el proyecto se deben negociar y registrar el compromiso de todos los involucrados (clientes, equipo de trabajo) con los requisitos.

- Administrar los cambios a los requisitos: se deben administrar los cambios a los requisitos conforme vayan evolucionando en el proyecto.
- Mantener la trazabilidad de los requisitos: respetar las dependencias de cada requisito con otros requisitos o artefactos.
- Identificar inconsistencias entre el trabajo del proyecto y los requisitos: deben realizarse revisiones con el fin de identificar las inconsistencias entre los planes de proyecto, los productos de trabajo, los requisitos y sus cambios (34).

Se puede determinar que CMMI propone la forma de validar que los requisitos estén correctos, orienta su documentación, lo cual es vital si se tiene en cuenta que esta información es la que va a sustentar el correcto desarrollo de la versión en curso y va a constituir el punto de partida para futuras versiones del producto. También propone el almacenamiento de las dependencias de cada requisito, lo cual es muy importante a la hora de hacer la planificación del proceso de desarrollo del *software*, así como para el control de versiones o cambios sobre los requisitos.

1.4.2 Proceso Racional Unificado

Posee una disciplina llamada “Requisitos” que brinda una guía para encontrar, organizar, documentar, y seguir los cambios de los requisitos funcionales y restricciones. Utiliza una notación de casos de uso y escenarios para representarlos. Los mismos son divididos en dos grupos: los requisitos funcionales que representan la funcionalidad del sistema y los requisitos no funcionales que representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica. Para capturarlos se propone conocer las opiniones de todos los interesados en el proyecto, no solo de los usuarios finales, y anotar todas sus peticiones. A partir de ellas hay que descubrir lo que necesitan y expresarlo en forma de requisitos (34).

Pasos para la identificación:

- Enumerar los requisitos candidatos: ideas del cliente, usuario, analista y desarrolladores.

- Comprender el contexto del sistema: se requiere un conocimiento detallado del contexto en el que se emplaza el sistema.
- Capturar los requisitos funcionales: la técnica inmediata para identificar los requisitos del sistema se basa en los casos de uso (para el usuario es el modo de utilizar el sistema).
- Capturar los requisitos no funcionales: los requisitos no funcionales especifican propiedades del sistema (36).

Cada requisito tiene un conjunto de características y a la vez las características poseen un conjunto de estados:

- Estado (propuesto, aprobado, incluido o validado).
- Coste estimado de implementación (en término de tipos de recursos y horas-persona).
- Prioridad (crítico, importante o secundario).
- Nivel de riesgo asociado a la implementación del requisito (crítico, significativo u ordinario).

A partir de lo anterior se puede determinar que el Proceso Racional Unificado (RUP, según sus siglas en inglés), propone mantener una lista de ideas (brindadas por el cliente y por el equipo de desarrollo para que el sistema pueda cumplir con las necesidades del usuario final), que se consideran como un conjunto de requisitos candidatos que se pueden implementar en la presente o futura versión del sistema. Este control es vital para tener una planificación de todo lo que se piensa hacer. Trata dos tipos de requisitos: funcionales y no funcionales, de los cuales propone almacenar algunas características, entre las que sobresale la prioridad (crítico, importante o secundario) que define cuán importante es el requisito para el cumplimiento de las expectativas del cliente y por ende cuán necesaria es su implementación y el nivel de riesgo asociado a la implementación del requisito (crítico, significativo u ordinario) que da una idea de la complejidad que el desarrollo de dicha funcionalidad conlleva.

1.4.3 Programación al extremo

Las historias de usuario son la técnica utilizada en Programación al extremo (XP, por sus siglas en inglés) para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Las escriben los propios clientes, tal y como ven ellos las necesidades del sistema. Debido a esto, el cliente es considerado parte del equipo y como tal, debe estar presente durante todo el proceso de desarrollo de *software*, constituyendo el pilar fundamental en el proceso de identificación de requisitos.

Respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. En muchos casos solo se propone utilizar un nombre y una descripción (37) o solo una descripción (38), mas quizás una estimación de esfuerzo en días (39). Beck en su libro (40) presenta un ejemplo de ficha (*customer story and task card*) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado, cosas por terminar y comentarios.

Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración (35). XP maneja el término “cosas por terminar”, que como su nombre indica son todos aquellos requisitos que no han sido desarrollados completamente o que aún no se han comenzado a implementar. Para esta metodología es de vital importancia la descripción de cada uno de ellos, pues de esta depende el entendimiento por parte del programador de qué se quiere hacer. La misma va a estar tan detallada como sea necesario y posteriormente se va a dividir en tareas de programación de acuerdo a su complejidad. XP propone también, según algunos autores, incorporar para cada requisito la prioridad técnica y del cliente, una referencia a otro requisito previo y el riesgo.

1.4.4 Scrum

Es una metodología de desarrollo de *software* muy simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto (41).

El primer paso en Scrum es cuando el cliente explica su visión del producto. Eventualmente esta visión se ve envuelta en un refinamiento y se transforma en una lista de prioridad en dependencia de la importancia que tenga cada una de las funcionalidades que el cliente desee. Esta lista se llama Pila del Producto y es la guía del proceso durante todo el ciclo de vida del *software*.

El subconjunto de dicha lista que se destina para la versión actual es conocido como la Pila de Liberación y en general, este documento es el foco de atención principal para el propietario del producto. La Pila se actualiza continuamente por el propietario para reflejar los cambios en las necesidades del cliente, nuevas ideas o puntos de vista, movimientos de la competencia, obstáculos técnicos que aparecen, entre otros elementos de interés. El equipo proporciona estimaciones de los esfuerzos requeridos para cada elemento de la Pila y con esta información el propietario es responsable de la asignación de una estimación del valor para el negocio de cada elemento individual. Con estas dos estimaciones (esfuerzo y valor) y tal vez con las de riesgo adicionales, el dueño del producto prioriza los elementos para maximizar la elección de los que posean un alto valor y se puedan realizar con poco esfuerzo. Estas estimaciones pueden ser renovadas en cada iteración en consonancia con el aumento del conocimiento del equipo; en consecuencia, se trata de una actividad de repriorización continua de los elementos de la Pila del Producto. Dichos elementos pueden variar significativamente en tamaño o esfuerzo, los más grandes se desglosan en elementos más pequeños durante el refinamiento de la Pila, y los más pequeños pueden ser consolidados (42).

Como se puede apreciar, en Scrum se almacenan los requisitos en la Pila del Producto, se vaya a implementar en la presente versión o no. De cada uno de ellos se almacena la estimación del

valor para el negocio y del esfuerzo; con estas dos variables se determina la prioridad de dicho requisito, la que guía todo el proceso de desarrollo del producto, pues es la que define el orden en que se implementan las funcionalidades.

1.4.5 Desarrollo Basado en Funcionalidades

Es un enfoque ágil para el desarrollo de sistemas. Dicho enfoque no hace énfasis en la obtención de los requerimientos, sino en cómo se realizan las fases de diseño y construcción. Además, hace énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del proyecto.

Está compuesto por 5 fases:

- Desarrollar un modelo general: los expertos del dominio ya tienen una idea del contexto y los requerimientos del sistema extraído de entrevistas con el cliente y de la comprensión del dominio del problema.
- Construir una lista de funcionalidades: con los requerimientos existentes se construye una lista de funcionalidades.
- Planificación por funcionalidades: la lista de funcionalidades es ordenada basándose en la prioridad y en las dependencias entre cada funcionalidad.
- Diseño de funcionalidades: se diseñan las funcionalidades.
- Construcción de funcionalidades: se construyen las funcionalidades.

Se puede concretar que el Desarrollo Basado en Funcionalidades (FDD, por sus siglas en inglés) propone que la lista de funcionalidades (que es su manera de tratar los requisitos) esté ordenada sobre la base de la prioridad y de la dependencia entre cada una de ellas, esto hace más ágil el proceso pues el orden de las funcionalidades indica en qué momento deben implementarse.

1.4.6 Nova - OpenUp

Es una metodología de desarrollo de distribuciones GNU/Linux, que se enfoca principalmente en el ciclo de vida de la distribución GNU/Linux Nova. Abarca todo el ciclo de vida de proyectos que hacen productos de este tipo, mediante la ejecución de un conjunto de disciplinas compuestas por actividades que se relacionan entre sí, roles y artefactos (2). Posee una disciplina llamada "Requisitos" donde proporciona las tareas necesarias para llevar a cabo la obtención, análisis, especificación, verificación, traceo, administración y validación de los requisitos de la distribución. Esta disciplina posee una actividad titulada "Identificar requisitos de la distribución GNU/Linux", cuyo objetivo es capturar los requisitos funcionales y no funcionales de la misma, los cuales se obtienen de:

- Los resultados de la fundamentación teórica de las distribuciones GNU/Linux que cumplan con los criterios de selección definidos.
- Las necesidades de los principales involucrados.
- Los reportes que se obtienen de los procesos de migración.
- La opinión de los diferentes usuarios que pertenecen a la comunidad.
- Las exigencias de clientes específicos en el caso que se quiera hacer una personalización de la distribución.
- Los paquetes de aplicaciones mínimos que necesita el sistema operativo base para su funcionamiento.
- La determinación del sistema anfitrión sobre el cual se va a desarrollar la distribución.

Dentro de esta disciplina también se encuentra la actividad "Detallar los requisitos de la distribución GNU/Linux", la cual consiste en detallar los requisitos funcionales, dejando claras las características de los mismos, su prioridad y complejidad. Esta disciplina propone establecer la trazabilidad de los requisitos, para lo cual es necesario poseer información acerca de las

dependencias entre ellos.

Se concluye que la metodología Nova - OpenUp propone una descripción para cada uno de los requisitos, de forma tal que el programador pueda tener una explicación detallada de lo que se quiere que realice cada uno, además de la prioridad y complejidad que poseen, todo esto da una idea de la importancia que tienen para el negocio. De igual manera se propone el almacenamiento de las dependencias que puedan poseer unos con otros para lograr establecer el orden en que se deben implementar los mismos.

1.4.7 Selección de las características para la identificación de requisitos

A raíz del estudio realizado se define que, de la misma manera que en la práctica los requisitos son extraídos de lo que el usuario espera para el negocio, en el presente problema y extrapolando lo anteriormente planteado, es necesario solicitarle a las aplicaciones que se encuentran alojadas en el repositorio de código fuente y binario, información acerca de los requisitos que quedaron por implementar. Esta información debe ser accesible si se tiene en cuenta que tanto CMMI en sus políticas, como la mayoría de las metodologías, proponen de una manera u otra, el almacenamiento de todos los requisitos, incluyendo los que no se van a implementar en la versión que se esté desarrollando.

Analizando las características que las metodologías proponen almacenar de cada requisito, en la presente propuesta de solución, se determinan los datos y características que deben poseer:

- Nombre del requisito.
- Descripción del requisito.
- Tipo de requisito (funcional, no funcional).
- Prioridad (baja, media, alta).
- Dependencias con otros requisitos.

Los requisitos se ordenan teniendo en cuenta las dependencias con otros requisitos y la prioridad

que posean. En ocasiones, para la implementación de un requerimiento, es necesario haber implementado otro previamente. De igual forma, varios requisitos se pueden implementar en un mismo momento, pero la prioridad es la que determina cuáles son más urgentes para el negocio. Esto permite conocer su significado y en qué orden deben ser implementados.

1.5 Repositorios de códigos fuente y binarios

Según varias fuentes de información, se pueden encontrar disímiles definiciones para repositorio:

- Un repositorio es un lugar en internet donde se almacena información, en el caso de los repositorios Linux esta información son programas (44).
- El repositorio es a todos los efectos un archivo ordenado donde son almacenados los paquetes Debian (sean estos paquetes binarios o fuente) en modo bien organizado, con una estructura bien definida y constantemente actualizados (45).
- Es un sitio centralizado donde se guarda información y archivos de los paquetes (imágenes, librerías, programas, código fuente) que se pueden instalar en Linux (46).

A partir de lo anterior se define que un repositorio de código fuente y binario es un sitio en la red donde se almacena el código fuente de las aplicaciones, así como binarios que son los que se instalan en el sistema operativo del usuario.

Al conjunto de datos que posee cada una de estas aplicaciones se le denomina paquete, que no es más que un archivo comprimido que tiene una estructura definida, lo que hace que las herramientas de gestión de *software* del sistema puedan ejecutarlo para realizar compilaciones, instalaciones y eliminaciones de los archivos de configuración del sistema, también actualizaciones e instalaciones de nuevas aplicaciones y funcionalidades, de una forma segura y centralizada (46).

Por su parte Torrelaguna expresa que un paquete es un archivo comprimido que contiene información del producto, archivos de programa, bibliotecas, íconos, documentación y scripts de

configuración (47).

Por lo que se puede definir que un paquete de *software* es un archivo comprimido con una estructura determinada que contiene la información necesaria de una aplicación para que la misma pueda ser instalada, eliminada o actualizada en el sistema operativo del usuario.

1.5.1 ¿Qué almacena un repositorio de cada paquete que posee?

Objeto: unidades discretas de información en forma digital, que se clasifican en tres tipos: archivo (*file*), representación (*representation*) y cadenas de bits *(bitstream*). El objeto archivo es tal cual entendemos normalmente, por ejemplo, un ejecutable. El objeto representación es el conjunto de todos los archivos que se necesitan para representar la entidad intelectual (por ejemplo, una aplicación). Los objetos cadenas de bits son subconjuntos de archivos con propiedades útiles a la preservación. En el ejemplo de la aplicación, el archivo ".jpeg" del logo puede tener sus propios identificadores. La información que se puede registrar en los objetos incluye: un identificador, la integridad, el tamaño, información sobre la creación, el entorno, el soporte y la relación con otros objetos y otros tipos de entidades.

Eventos: agrega información sobre acciones que un agente, o varios, lleva a cabo sobre los objetos de los repositorios, por ejemplo: el identificador del acontecimiento (no repetible), el tipo (creación y migración), la fecha de ocurrencia del evento, la descripción y el resultado codificado del acontecimiento, así como los agentes.

Agentes: pueden ser personas, organizaciones o aplicaciones de *software* con actividades o responsabilidades en los eventos. Se aconseja como información: un identificador único, el nombre del agente y su tipo (por ejemplo, persona) (48).

1.6 Actividades dentro de un repositorio

El proceso puede iniciarse cuando el productor suministra el recurso (aplicación), al que luego se

le hacen una serie de modificaciones para ser almacenado en el repositorio (la aplicación se convierte en un paquete). El flujo puede continuar cuando el consumidor busca una información en el sistema, que es entregada. Por último el administrador puede modificar o eliminar un paquete del repositorio (48).

De lo anterior, se puede definir que las actividades que se realizan dentro de un repositorio son:

- Suministro: cuando se le propone nueva información al repositorio y esta es insertada.
- Modificación: cuando se modifica la información dentro del repositorio.
- Consulta: cuando se le solicita alguna información al repositorio y es devuelta la misma.
- Eliminación: cuando se elimina alguna información del repositorio.

1.6.1 ¿Cómo se organizan los elementos en un repositorio?

Las comunidades y colecciones sirven a los fines administrativos para organizar los elementos dentro del repositorio, aunque en muchos de ellos se los considera como meta-datos, dado que representan información estructural sobre los elementos (48)**.**

No se detectó la existencia de ninguna actividad que se encargue de la identificación de requisitos, ni relacionada con la ingeniería de requisitos, dentro de los repositorios.

Los repositorios almacenan paquetes, los cuales están compuestos por una gama de información asociada al mismo, además de la propia aplicación que el paquete constituye (por ejemplo, identificador, la integridad, el tamaño, información sobre la creación, sobre el entorno, el soporte y la relación con otros objetos y tipos de entidades).

Luego del análisis de la estructura de los repositorios de códigos fuente y binarios, sus actividades y el contenido que manejan, se detectó que entre las actividades que posee no se encuentra la identificación de requisitos. Debido a esto, se determina que la incorporación de un archivo que forme parte de los datos de los paquetes, es una manera efectiva de proporcionar información acerca de los requisitos que le faltan por implementar a los mismos. De esta manera,

al solicitarle al repositorio los requisitos que le faltan por desarrollar a las aplicaciones que almacena, su respuesta sería a través del archivo propuesto que poseería cada una.

1.7 Conclusiones del capítulo

- El estudio de la manera en que se realiza la identificación de requisitos de las aplicaciones que almacenan los repositorios en CESOL, permitió determinar que el mecanismo utilizado es ineficiente y que no existe una herramienta definida que cumpla ese objetivo.
- El análisis de lo que proponen las metodologías de desarrollo de *software* para identificar requisitos, permitió definir la forma en que se debe hacer en el sistema a desarrollar, así como los datos que son necesarios almacenar de cada uno de ellos, para que su significado y planificación sean claros.
- Para lograr la integración del sistema propuesto con la aplicación “Metodología para el desarrollo de distribuciones GNU/Linux”, se decidió realizar un complemento, utilizando la metodología ágil SXP, por ser un proyecto de corta duración con un equipo de desarrollo pequeño.
- Se definió utilizar el sistema de gestión de contenido Drupal, con la utilización de los lenguajes de programación HTML, PHP, Java Script y *Python*, además del uso del sistema gestor de base de datos PostgreSQL.
- El estudio de la estructura, actividades e información que poseen los repositorios de códigos fuente y binarios, permitió definir la manera en que se puede realizar la identificación de requisitos a partir del mismo, determinándose que la incorporación de un archivo en las carpetas de datos de los paquetes con los requisitos que faltan por implementar, es la manera más eficiente teniendo en cuenta las prácticas que proponen las metodologías.

Capítulo 2: Requisitos y arquitectura de la aplicación para la identificación de requisitos en un repositorio de código fuente y binario

En el presente capítulo se define la manera en que se construye el sistema, o sea, se plasman sus características a través de los requisitos funcionales, no funcionales y las historias de usuario que especifican cada requisito funcional, además se describe la propuesta de solución donde se incluye la arquitectura del *software* para conocer la manera en que se estructura, los patrones de diseño a utilizar, los diagramas de clases, el diagrama de paquetes y por último el diagrama de despliegue.

2.1 Descripción de la aplicación

2.1.1 Modelo de dominio

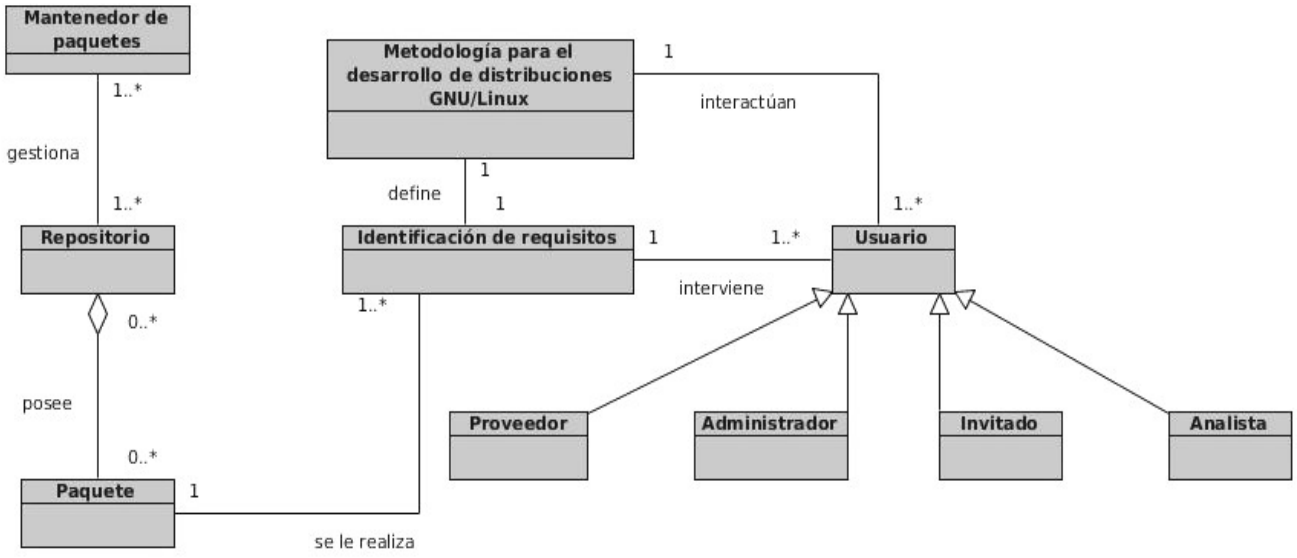

Figura 1: Modelo de dominio.

A continuación se detallan los diferentes objetos y relaciones que conforman el proceso:

Mantenedor de paquetes: constituye un rol propuesto por la metodología de desarrollo de *software* Nova – OpenUp, que se encarga de la gestión de los paquetes del repositorio.

Repositorio: se refiere a los repositorios de códigos fuente y binarios, está compuesto por los paquetes del sistema y por el código fuente de las aplicaciones de código abierto que posee.

Paquetes: constituyen aplicaciones informáticas con la información necesaria para que puedan ser instaladas en el sistema operativo del usuario.

Identificación de requisitos: es la actividad que se encarga de determinar cuáles son las funcionalidades que debe poseer el sistema para cumplir con las expectativas del usuario y con las especificidades del negocio.

Metodología para el desarrollo de distribuciones GNU/Linux: es una aplicación informática cuya misión es apoyar la metodología de desarrollo de *software* Nova - OpenUp.

Usuario: es la persona que trabaja con la herramienta y que su interacción con el sistema depende de sus responsabilidades dentro del negocio.

Proveedor: usuario que se encarga de proveer requisitos a un sistema.

Analista: usuario que se encarga de realizar las tareas del proceso de gestión de requisitos.

Invitado: usuario que navega en la aplicación sin haberse registrado aún.

Administrador: usuario que posee control total sobre la aplicación.

2.2 Condiciones para el funcionamiento de la aplicación a desarrollar

La aplicación a desarrollar es funcional para repositorios de códigos fuente. La misma identifica cuál de esos repositorios es el que utiliza el SO a partir de los datos configurados en el sources.list[6] e incide sobre él. Los paquetes de código fuente poseen una carpeta en su interior llamada “debian”, donde se almacena información necesaria acerca del paquete. En la presente

6 sourse.list: archivo que se encuentra en /etc/apt/source.list donde se configuran los repositorios.

investigación se propone incorporar un nuevo archivo a esta carpeta llamado "*To_Do"* en el cual se almacenen los requisitos que le faltan por implementar a dicho paquete. Este archivo se estructura de la siguiente manera:

Nombre: nombre del requisito.

Descripción: descripción del requisito.

Tipo: funcional o no funcional.

Prioridad: alta, media o baja.

Dependencias: nombre de otros requisitos que deben estar previamente implementados.

A este grupo de información se le denomina "bloque", aparecen tantos como requisitos sin implementar y se evita el uso de caracteres extraños.

La aplicación se apoya en estos archivos y lee la información que los mismos contienen, por lo que el cumplimiento de la estructura propuesta es de vital importancia para el correcto funcionamiento de la presente propuesta de solución.

2.3 Requisitos funcionales y no funcionales

Tabla 1: Requisitos funcionales y no funcionales.

Código	Descripción de los requisitos funcionales	Prioridad
RF1	Mostrar todos los paquetes teniendo en cuenta sus clasificaciones.	alta
RF2	Buscar requisitos de paquetes seleccionados.	alta
RF3	Buscar paquete.	baja
RF4	Mostrar requisitos que faltan por implementar.	alta
RF5	Guardar requisitos.	alta
RF6	Exportar requisitos.	media
RF7	Cancelar resultado.	baja
RF8	Actualizar base de datos.	alta
RF9	Crear archivo *To_Do*.	alta

RF10	Mostrar requisitos guardados.	alta
Descripción de los requisitos no funcionales		
Restricciones de *software*		
RNF1	Características de *software*: • Sistema operativo: Nova 2011 hasta Nova 2015. • Navegador web: Mozilla Firefox 2.0 hasta 38.0. • Servidor web: Apache 2.0 hasta 2.2. • Versión de php 5.0. • Sistema gestor de bases de datos: PostgreSQL 8.4 hasta 9.1. • Paquete de conexión a base de datos para *python*: *python*-psycopg2. • Paquete de descompresión de paquetes fuentes: dpkg-dev.	
RNF2	Características de *hardware* (PC Cliente): • 512 Mb de memoria RAM o superior. • 80 Gb de disco duro o superior. • Microprocesador: Intel Pentium IV o superior.	
RNF3	Características de *hardware* (servidores): • Servidor web Apache: el servidor debe contar como mínimo con las siguientes características: 4GB de memoria RAM, un microprocesador Intel Core 2 Duo y un disco duro de 250GB. • Servidor Base de datos: el servidor debe contar como mínimo con 2GB de memoria RAM, un microprocesador Intel Core 2 Duo y un disco duro de 160GB.	
Restricciones de diseño e implementación		
RNF4	Lenguajes de programación: • PHP 5. • *Python* 2.7.6. • HTML 5.	

	• CSS 3.
RNF5	La interfaz visual debe mantener el estilo y diseño de la aplicación “Metodología para el desarrollo de distribuciones GNU/Linux”.
RNF6	Herramientas de desarrollo : • CMS Drupal 7.26. • Visual Paradigm 8.0.
Componentes adquiridos	
RNF8	Uso de licencias libres y flexibles que hayan sido aprobadas por la Fundación de *Software* Libre: • GNU/GPL para el CMS Drupal. • BSD[7] de PosgreSQL. • PHP *License*.

2.4 Propuesta de la aplicación para la identificación de requisitos a partir de un repositorio de aplicaciones

A continuación se describe la estructura de la aplicación para una mejor comprensión por parte del usuario.

Al abrir el sistema se pueden apreciar tres regiones: la superior, compuesta por un buscador para facilitar el acceso al paquete deseado; la izquierda, compuesta por cada una de las clasificaciones en las que se pueden agrupar los paquetes del repositorio y la derecha, donde se muestran los paquetes que se encuentran albergados actualmente en el mismo. Al dar clic a las clasificaciones, se puede acceder al grupo que se enmarca dentro de la misma y dicha información se muestra en la región derecha. El usuario puede seleccionar las aplicaciones de las cuales quiere conocer los requisitos que le faltan por implementar y al presionar el botón “Buscar requisitos” aparece una nueva región, compuesta por un campo de texto donde se

7 BSD: licencia que permite usar el código fuente del *software* libre en *software* privativo.

muestra el resultado de la búsqueda y tres botones: “Guardar”, que permite almacenar la información en la base de datos; “Exportar”, que posibilita exportar dicha información hacia un archivo externo y “Cancelar” que redirecciona nuevamente hacia el estado inicial de la aplicación. También aparecen tres botones llamados “Actualizar BD”, “Crear *To_Do*” y “Mostrar revisados” en la primera vista descrita. El primero permite almacenar los paquetes que se encuentran en el repositorio en la base de datos para que las búsquedas realizadas por la aplicación sean más rápidas, el segundo posibilita la creación del archivo *To_Do* con las características definidas en la presente investigación y el tercero muestra los requisitos que hayan sido guardados por el usuario en la base de datos.

2.5 Historias de usuario

La manera de describir requisitos en la metodología SXP es mediante la realización de las historias de usuario. Durante la fase de Planificación-Definición se establecieron 10 historias de usuario, las cuales responden a cada una de las funcionalidades del sistema. A continuación se describen algunas de prioridad alta.

Tabla 2: Historia de usuario. Mostrar todos los paquetes teniendo en cuenta sus clasificaciones.

Historia de usuario	
Número: 1	**Nombre de historia de usuario:** Mostrar todos los paquetes teniendo en cuenta sus clasificaciones.
Usuario: Manuel Enrique Peiso Cruz	**Iteración Asignada:** 1
Prioridad: alta	**Puntos Estimados:** 2,5 semanas
Riesgo en Desarrollo: N/A	**Puntos Reales:** 2,5 semanas
Descripción: el usuario debe poder ver todos los paquetes que están almacenados en el repositorio, pero también debe tener la opción de buscarlos por clasificación. Las clasificaciones son: amateur radio, comunicación, multiplataforma, base de datos, depuradores, desarrollo, documentación, edición, educación, electrónico, correo electrónico, paquetes integrados, paquetes fuente, entorno de escritorio gnome, sistema estático gnu r, datos de	

introspección gobject, juegos, entorno de escritorio gnustep, gráficos, lenguaje de programación haskell, internacionalización y localización, lenguaje interpretado, lenguaje de programación java, entorno de escritorio kde, lenguaje de programación ruby, ciencia, consola, administración del sistema, auditoría teX, utilidades, kernel y módulos, librerías, librerías de desarrollo, librerías antiguas, lenguaje de programación lisp, localización, matemática, metapaquetes, gráficos diversos, basados en textos diversos, infraestructura mono/cli, multimedia, red, grupo de noticias, lenguaje de programación ocalm, lenguaje de programación php, lenguaje de programación perl, lenguaje de programación *python*, control de versiones, *software* de video, servidores web, procesadores de texto, *world wide web*, entorno de escritorio xfce, entorno zope/plone.
Observaciones: cuando el usuario abre la aplicación, por defecto sale el listado con todos los paquetes albergados en el repositorio, pero también tiene la posibilidad de seleccionar una clasificación determinada para ver los que pertenecen a la misma.
Prototipo de interfaz: ver Anexo 2.

Tabla 3: Historia de usuario. Buscar requisitos de paquetes seleccionados.

Historia de usuario	
Número: 2	**Nombre de historia de usuario**: Buscar requisitos de paquetes seleccionados.
Usuario: Manuel Enrique Peiso Cruz	**Iteración Asignada:** 1
Prioridad: alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: N/A	**Puntos Reales:** 1 semana
Descripción: el usuario debe poder buscar de los paquetes seleccionados los requisitos que le faltan por implementar.	
Observaciones: el usuario para poder ejecutar la funcionalidad del botón Buscar requisitos debe haber seleccionado al menos un paquete.	
Prototipo de interfaz: ver Anexo 3.	

Tabla 4: Historia de usuario. Mostrar requisitos que faltan por implementar.

Historia de usuario	
Número: 4	**Nombre de historia de usuario**: Mostrar requisitos que faltan por implementar.
Usuario: Manuel Enrique Peiso Cruz	**Iteración Asignada:** 2
Prioridad: alta	**Puntos Estimados:** 1 semana
Riesgo en Desarrollo: N/A	**Puntos Reales:** 1 semana

Descripción: el usuario debe poder visualizar los requisitos que le faltan por implementar de las aplicaciones que seleccionó y revisó previamente.
Observaciones: esta información aparece en un campo no editable.
Prototipo de interfaz: ver Anexo 4.

Tabla 5: Historia de usuario. Guardar requisitos.

Historia de usuario	
Número: 5	**Nombre de historia de usuario**: Guardar requisitos.
Usuario: Manuel Enrique Peiso Cruz	**Iteración Asignada:** 2
Prioridad: alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: N/A	**Puntos Reales:** 2,5 semanas
Descripción: el usuario debe poder guardar la información devuelta por el sistema acerca de los requisitos que faltan por implementar en los paquetes revisados.	
Observaciones: esta información se almacena en la base de datos “Metodología” para futuras consultas.	
Prototipo de interfaz: ver Anexo 5.	

2.6 Descripción de la Arquitectura de *software* y los patrones de diseño

2.6.1 Arquitectura de *software*

La arquitectura del *software* de un programa es la estructura del sistema, que incluye los componentes del *software*, sus propiedades visibles externamente y las relaciones entre ellos (4). El complemento a implementar utiliza una variante del patrón arquitectónico de N-capas, el cual organiza el sistema en capas, cada una de las cuales proporciona un conjunto de servicios. Esta aproximación soporta el desarrollo incremental, a medida que se implementa un nivel, algunos de los servicios que proporciona pueden estar disponibles para los usuarios. Las capas internas brindan facilidades básicas, como gestión de ficheros, que son requeridas por todos los niveles. Los servicios que necesita un usuario de nivel superior pueden, por lo tanto, tener que atravesar las capas adyacentes para tener acceso a los proporcionados por los niveles inferiores (28). Los

componentes de cada capa se comunican con los de otras a través de interfaces.

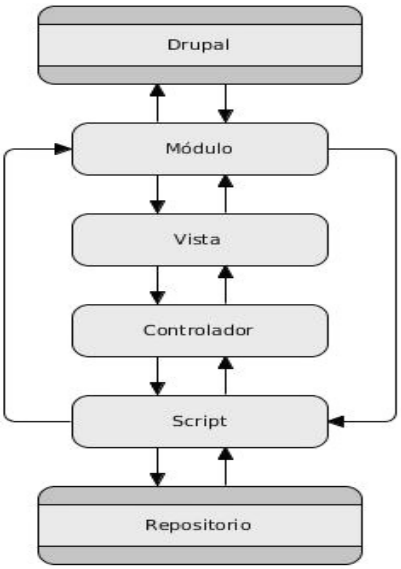

Figura 2: Patrón arquitectónico N-capas, estilo 4-capas.

Descompone los servicios de forma que las interacciones ocurren solo entre capas vecinas. Precisamente se plantea que se usa una variante de este patrón porque en la presente propuesta de solución, no se cumple estrictamente dicha propiedad. Como se puede apreciar en la Figura 2, la capa Módulo es capaz de comunicarse con la capa Script y viceversa a pesar de no ser adyacentes, esto se realiza debido a la necesidad de que haya una interacción continua entre ambas, algo que resultaría muy engorroso si se hiciera de la manera tradicional.

El sistema está compuesto por cuatro capas (ver figura 2):

- **Módulo:** se encuentran los archivos necesarios que utiliza Drupal para reconocer al sistema como un módulo.
- **Vista:** se encuentran todas las vistas del sistema.
- **Controlador:** se almacenan las clases controladoras que son las encargadas de realizar las operaciones básicas que debe cumplir el sistema.

- **Script:** se encuentran scripts que se encargan de la comunicación directa con el repositorio y de la creación de ficheros que van a ser utilizados por las capas superiores.

2.6.2 Patrones de diseño

Brad Appleton define un patrón de diseño de la siguiente manera: “Un patrón es una semilla de conocimiento que tiene un nombre y transporta la esencia de una solución probada a un problema recurrente dentro de cierto contexto en medio de intereses en competencia” (49).

Pressman por su parte expresa que un patrón de diseño describe una estructura que resuelve un problema de diseño particular dentro de un contexto específico en medio de “fuerzas” que pueden tener un impacto en la manera en que se aplica y utiliza el patrón (3).

De otra manera se puede decir que un patrón de diseño es una estructura de diseño probada con anterioridad para problemas recurrentes en el contexto que se esté trabajando.

Se utilizan los siguientes:

Singleton

Es un patrón de diseño que se utiliza para garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella (4).

En la aplicación es imprescindible que exista solamente una instancia de los ficheros que se van a utilizar para leer la información almacenada en el repositorio. Esta información va a ser accedida desde un solo punto del sistema (la interfaz principal del mismo) permitiendo un acceso controlado a la única instancia. Por ejemplo, al ejecutar el script que se encarga de leer el nombre de todos los paquetes almacenados en el repositorio, se va a generar un archivo donde se almacenan los mismos, de este archivo es necesario tener una única instancia que se va a utilizar para almacenar en la base de datos la información contenida en él.

Facade

Es un patrón de diseño de tipo estructural, usado con el objetivo de proporcionar una interfaz

unificada para un conjunto de interfaces en un subsistema, haciéndolo más fácil de utilizar. Se aplica cuando se quiere proporcionar una interfaz sencilla para un subsistema complejo y se desee desacoplar un *software* de sus clientes y de otros subsistemas, haciéndolo más independiente y portable (4).

En la presente propuesta de solución se utiliza en la interfaz principal que se va a encargar de unificar toda la información que brinde el subsistema haciéndola más centralizada, manipulable y permitiendo que la misma delegue las peticiones de los clientes en los objetos del subsistema. De igual manera, al utilizar la estructura de un módulo de Drupal, el sistema va a tener un funcionamiento independiente al de la herramienta “Metodología para el desarrollo de distribuciones GNU/Linux” a la cual se va a integrar.

Iterator

Es un patrón de diseño de comportamiento que proporciona una forma de acceder secuencialmente a los elementos de un agregado sin revelar su representación interna (4).

En la presente propuesta de solución se usa para el acceso a los elementos del repositorio, recorriéndolos y accediendo a su contenido. El sistema va a proporcionar una interfaz uniforme que no incluye las operaciones de iteración, haciendo parecer sencillo para el usuario una serie de pasos complejos que el sistema necesita realizar como la descarga de los paquetes seleccionados y la lectura del archivo *To_Do*.

Mediator

Es un patrón de diseño de comportamiento que se utiliza para definir un objeto que especifique cómo interactúan un conjunto de objetos, evitando que tengan que conocerse entre ellos y permitiendo cambiar la interacción de forma independiente (4).

Se evidencia en las clases controladoras, que son las encargadas de definir la manera en que se van a relacionar los elementos de la capa Vista con los de la capa Script, los cuales, a pesar de no conocerse entre sí, se van a comunicar a través de los pasos definidos por la controladora.

2.7 Diagramas de clases

En el diagrama de clases del diseño del subpaquete “Mostrar paquetes” (ver Figura 3), se puede apreciar que la clase ToDo.module se encarga de ejecutar el script Precondiciones.py, a través del cual se crean los directorios necesarios para el funcionamiento de la aplicación. Con esta información se manda a ejecutar la clase servidora SPMostrarPaquetes, que es la encargada de construir la página cliente CPMostrarPaquetes, esta última se compone de un formulario denominado HTMLFormMostrarPaquetes, en el cual se muestra un listado de checkbox con todas las aplicaciones que se encuentran almacenados en la base de datos. También se muestran las diferentes clasificaciones en las que se pueden agrupar los paquetes, que, al ser seleccionadas, se ejecutan mediante la controladora CControladoraTodos que es la que indica cuáles paquetes deben ser mostrados.

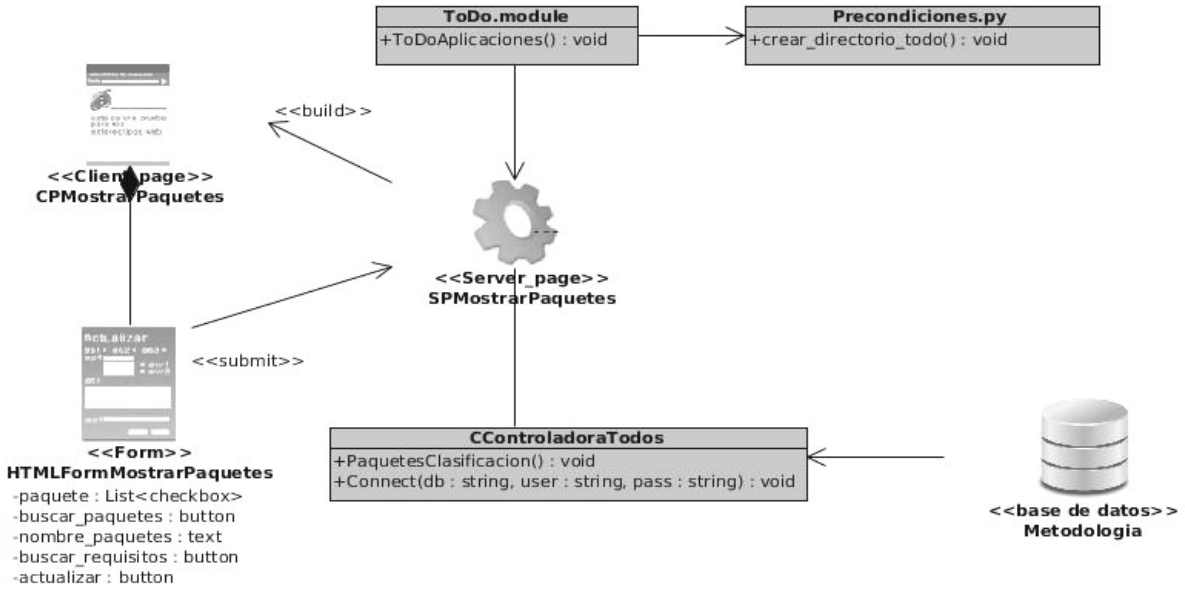

Figura 3: Mostrar paquetes.

En el diagrama de clases llamado “Buscar requisitos” (ver figura 4), la página servidora

SPBuscarRequisitos, utilizando la información mostrada por el subpaquete “Mostrar paquetes”, construye la página cliente CPBuscarRequisitos, compuesta por un formulario llamado HTMLFormBuscarRequisitos, en el cual se pueden seleccionar los paquetes que se deseen revisar en busca de los requisitos que le faltan por implementar y a través del botón “Buscar requisitos” mandar a realizar dicha operación. La clase controladora CControladoraTodos es la encargada de mandar a ejecutar el script DescargarArchivo.py, que se comunica directamente con el repositorio, explorando en cada uno de los paquetes seleccionados, si existe el archivo *To_Do.* En caso afirmativo almacena la información, de lo contrario muestra que no existe.

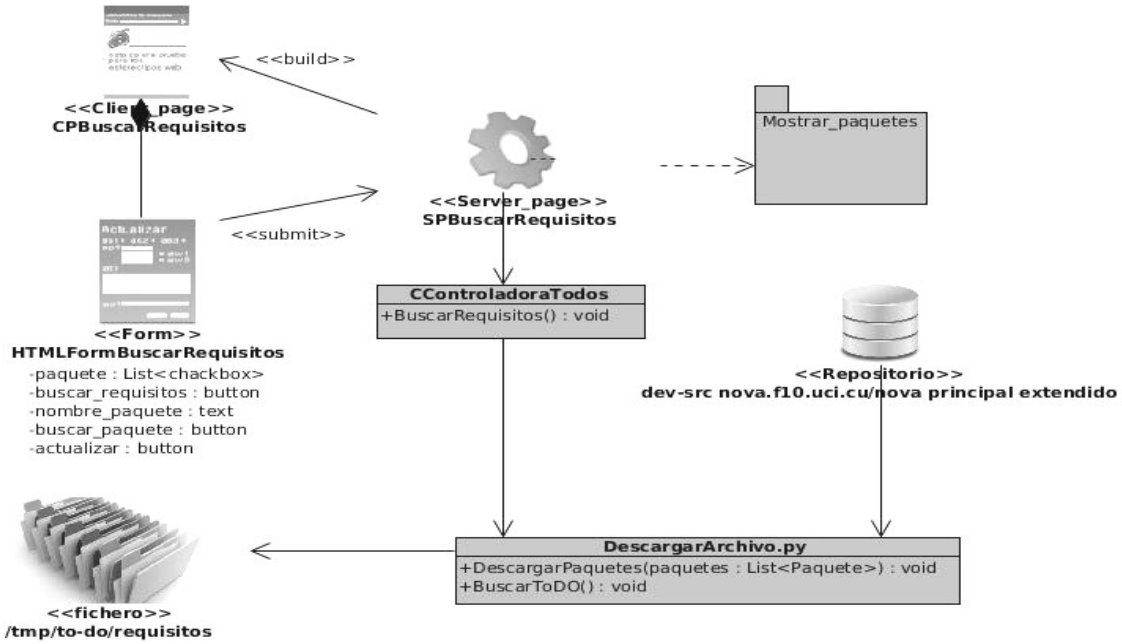

Figura 4: Buscar requisitos.

2.8 Diagrama de paquetes

Para lograr un mayor entendimiento del funcionamiento del sistema, se hace necesario describir su estructura física, esto se debe a que las mejoras que se le realicen posteriormente deben

estar acopladas a la estructura que se define, proporcionando uniformidad al sistema en general. En la siguiente figura se presenta el diagrama de paquetes correspondiente a la extensión a desarrollar, en el cual se evidencia la arquitectura seleccionada.

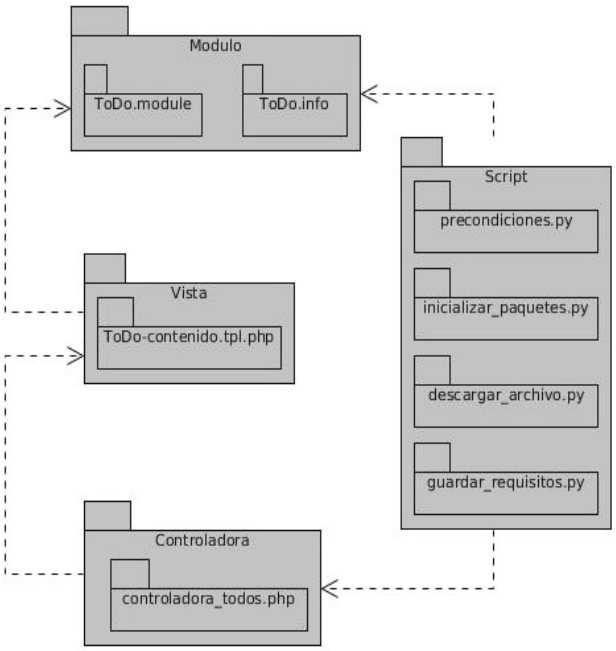

Figura 5: Diagrama de paquetes.

2.9 Modelo de despliegue

El diagrama de despliegue es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre dichos elementos. En la siguiente figura se muestra el diagrama correspondiente a la aplicación a desarrollar. En el mismo, la PC Cliente representa las computadoras de los usuarios que se conectan al sistema, las cuales realizan peticiones al servidor web mediante el protocolo HTTPS. Este servidor mantiene una conexión mediante el protocolo TCP/IP al servidor de bases de datos.

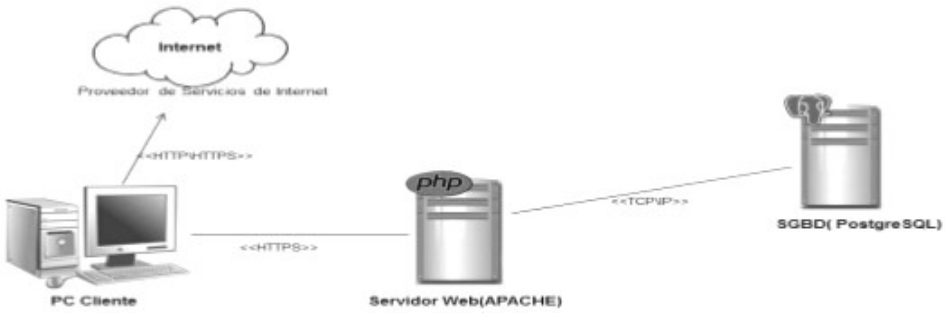

Figura 6: Diagrama de despliegue.

2.10 Conclusiones del capítulo

- A partir de la estructura que poseen los repositorios y los paquetes que contienen, se definió la incorporación de un archivo llamado *To_Do* en la carpeta debian de cada paquete fuente que se encuentre en el repositorio. La aplicación va a buscar la información almacenada en dicho archivo para darle respuesta a las peticiones del usuario.
- Se definió la información que debe almacenar el archivo *To_Do* y la manera en que debe encontrarse la misma, esta información fue definida bajo el término “bloque”.
- La comprensión de las necesidades del usuario, las características del proceso y las restricciones bajo las que el sistema debe funcionar, permitió definir los requisitos funcionales y no funcionales, arquitectura y patrones a utilizar.
- El entendimiento de lo que propone la metodología de desarrollo de *software* a utilizar, permitió describir las historias de usuario para una mejor comprensión de los requisitos funcionales, así como los diagramas de clases del diseño, el diagrama de paquetes y el diagrama de despliegue.

Capítulo 3: Implementación y evaluación de la aplicación para la identificación de requisitos en un repositorio de código fuente y binario

En el presente capítulo se implementan los componentes necesarios del complemento para la identificación de requisitos en un repositorio de código fuente y binario (*ToDo)*, la implementación se obtiene a partir de los resultados obtenidos en las fases anteriores.

Dentro de los principales objetivos a lograr se encuentra implementar los elementos de diseño en términos de “elementos de implementación” (ficheros fuentes, binarios y ejecutables), abriendo el camino a las pruebas, donde cada construcción generada es sometida a revisión en busca de no conformidades.

Siguiendo el trabajo con la metodología seleccionada, se presentan el plan de iteraciones y las tareas de ingeniería por cada historia de usuario, posteriormente se van a describir los casos de prueba de aceptación a los que fueron sometidas las funcionalidades del sistema en cada una de las iteraciones.

3.1 Plan de iteraciones de implementación

Una vez realizadas las historias de usuario, y estimado el esfuerzo que su desarrollo conlleva, se hace necesario realizar la planificación de las etapas de implementación del complemento. El plan de entrega está compuesto por tres iteraciones, las cuales fueron fragmentadas para así poder obtener un trabajo incremental.

Iteración 1

En la primera iteración se implementan las historias de usuario 1, 2 y 8, que constituyen el eje central de la estructura básica del complemento. Se obtiene una primera versión del producto que puede comenzar a probarse en busca de errores.

Iteración 2

En esta iteración se implementan las historias de usuario 4, 5, 9 y 10, por ser las de prioridad alta que faltan por desarrollar y que permiten la visualización y el almacenamiento de la información. También se corrigieron errores detectados en las funcionalidades implementadas en la iteración anterior. Se obtiene una segunda versión del producto que es puesta en manos del cliente para su aceptación, recopilar inconformidades y sugerencias.

Iteración 3

En la tercera y última iteración se implementaron las historias de usuario 3, 6, y 7, por ser las que poseen prioridad media y baja y que estaban destinadas a hacer el sistema más fácil de utilizar. De igual forma se corrigieron las inconformidades detectadas en la iteración anterior. Se obtuvo una versión con todas las funcionalidades que se espera el producto posea.

3.1.1 Duración de iteraciones

Tabla 6: Plan de iteraciones.

Iteración	Descripción de la iteración	Orden de las HU a implementar	Duración total
1	Desarrollo de las Historias de usuario (HU) de mayor importancia.	1, 2 y 8	5,5 semanas
2	Desarrollo de las HU de prioridad alta que aún no se han implementado.	4, 5, 9 y 10	6,5 semanas
3	Desarrollo de las HU de prioridad media y baja.	3, 6, 7	4,5 semanas

3.2 Tareas de ingeniería

Las tareas de ingeniería constituyen una definición detallada de las Historias de usuario (HU), facilitando el entendimiento del proceso de implementación. Cada HU puede contener una o más tareas en caso de necesitarlas, explicando paso a paso las acciones que se realizan en las

mismas. Permiten además organizar el proceso de implementación, así como conocer el grado de complejidad de cada HU, teniendo en cuenta la cantidad de tareas asociadas.

Tabla 7: Tarea de ingeniería. Diseñar el prototipo de interfaz principal del complemento.

Tarea de Ingeniería	
Número Tarea: 1	**Número de Historia de Usuario:** 1, 2, 3, 8, 9 y 10
Nombre de Tarea: Diseñar el prototipo de interfaz principal del complemento.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 02/02/2015	**Fecha Fin:** 05/02/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se diseña una interfaz que permita a los usuarios visualizar los paquetes que están alojados en el repositorio, seleccionar los que deseen revisar en busca de los requisitos que le faltan por implementar, buscar algún paquete en específico, almacenar en la base de datos los paquetes, crear el archivo *To_Do* con las características definidas y mostrar los requisitos que se hayan obtenido de revisiones anteriores.	

Tabla 8: Tarea de ingeniería. Implementar la funcionalidad Actualizar base de datos.

Tarea de Ingeniería	
Número Tarea: 2	**Número de Historia de Usuario:** 8
Nombre de Tarea: Implementar la funcionalidad Actualizar base de datos.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1,5 semana.
Fecha Inicio: 08/02/2015	**Fecha Fin:** 18/02/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Actualizar base de datos”, que permite guardar en la base de datos el nombre de los paquetes que están almacenados en el repositorio, así como sus clasificaciones.	

Tabla 9: Tarea de ingeniería. Estudiar aplicación aptitude.

Tarea de Ingeniería	
Número Tarea: 3	**Número de Historia de Usuario:** 1
Nombre de Tarea: Estudiar aplicación aptitude.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 05/02/2015	**Fecha Fin:** 08/02/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se estudia la aplicación aptitude para determinar la manera en que explora el repositorio y dónde guarda la información.	

Tabla 10: Tarea de ingeniería. Implementar la funcionalidad Mostrar paquetes.

Tarea de Ingeniería	
Número Tarea: 4	**Número de Historia de Usuario:** 1
Nombre de Tarea: Implementar la funcionalidad Mostrar paquetes.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1,5 semana.
Fecha Inicio: 18/02/2015	**Fecha Fin:** 28/02/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Mostrar paquetes”, que permite mostrar el nombre de todos los paquetes que se encuentran en el repositorio, así como sus clasificaciones.	

Tabla 11: Tarea de ingeniería. Implementar la funcionalidad Buscar requisitos.

Tarea de Ingeniería	
Número Tarea: 5	**Número de Historia de Usuario:** 2
Nombre de Tarea: Implementar la funcionalidad Buscar requisitos.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1 semana.
Fecha Inicio: 28/02/2015	**Fecha Fin:** 07/03/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Buscar requisitos”, que permite buscar en los paquetes alojados en el repositorio la información referente a los requisitos que faltan por implementar.	

Tabla 12: Tarea de Ingeniería. Diseñar prototipo de interfaz para mostrar requisitos que faltan por implementar, guardarlos, exportarlos y cancelar operación.

Tarea de Ingeniería	
Número Tarea: 6	**Número de Historia de Usuario:** 4, 5, 6 y 7
Nombre de Tarea: Diseñar prototipo de interfaz para mostrar requisitos que faltan por implementar, guardarlos, exportarlos y cancelar operación.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 07/03/2015	**Fecha Fin:** 10/03/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se diseña una interfaz que permita a los usuarios visualizar los requisitos que faltan por implementar en los paquetes seleccionados, guardarlos, exportarlos y cancelar la operación.	

Tabla 13: Tarea de ingeniería. Implementar la funcionalidad Mostrar requisitos que faltan por implementar.

Tarea de Ingeniería	
Número Tarea: 7	**Número de Historia de Usuario:** 4
Nombre de Tarea: Implementar la funcionalidad Mostrar requisitos que faltan por implementar.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 10/03/2015	**Fecha Fin:** 13/03/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Mostrar requisitos que faltan por implementar”, que permite visualizar los requisitos detectados en la búsqueda realizada.	

Tabla 14: Tarea de ingeniería. Estudiar librería psycopg2.

Tarea de Ingeniería	
Número Tarea: 8	**Número de Historia de Usuario:** 5
Nombre de Tarea: Estudiar librería psycopg2.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1 semana.
Fecha Inicio: 13/03/2015	**Fecha Fin:** 20/03/2015

Programador Responsable: Manuel Enrique Peiso Cruz.
Descripción: se estudia la librería psycopg2, que permite almacenar información en una base de datos usando el lenguaje de programación *python*.

Tabla 15: Tarea de ingeniería. Implementar funcionalidad Guardar en la base de datos.

Tarea de Ingeniería	
Número Tarea: 9	**Número de Historia de Usuario:** 5
Nombre de Tarea: Implementar funcionalidad Guardar en la base de datos.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1 semana.
Fecha Inicio: 20/03/2015	**Fecha Fin:** 27/03/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Guardar en la base de datos”, para poder almacenar la información detectada por la aplicación.	

Tabla 16: Tarea de ingeniería. Implementar funcionalidad Exportar información.

Tarea de Ingeniería	
Número Tarea: 10	**Número de Historia de Usuario:** 6
Nombre de Tarea: Implementar funcionalidad Exportar información.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 2 semanas.
Fecha Inicio: 27/03/2015	**Fecha Fin:** 10/04/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Exportar información”, que permite almacenar la información detectada en un archivo externo en formato PDF.	

Tabla 17: Tarea de ingeniería. Implementar funcionalidad Cancelar operación.

Tarea de Ingeniería	
Número Tarea: 11	**Número de Historia de Usuario:** 7
Nombre de Tarea: Implementar funcionalidad Cancelar operación.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 10/04/2015	**Fecha Fin:** 13/04/2015

Programador Responsable: Manuel Enrique Peiso Cruz.
Descripción: se implementa la funcionalidad “Cancelar operación”, que permite volver a la interfaz que muestra los paquetes del repositorio.

Tabla 18: Tarea de ingenería. Implementar funcionalidad Buscar paquete.

Tarea de Ingeniería	
Número Tarea: 12	**Número de Historia de Usuario:** 3
Nombre de Tarea: Implementar funcionalidad Buscar paquete.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 13/04/2015	**Fecha Fin:** 16/04/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementa la funcionalidad “Buscar paquete”, que permite buscar un paquete cuyo nombre ya es conocido.	

Tabla 19: Tarea de ingeniería. Diseñar prototipo de interfaz para crear archivo To_Do.

Tarea de Ingeniería	
Número Tarea: 13	**Número de Historia de Usuario:** 9
Nombre de Tarea: Diseñar prototipo de interfaz para crear archivo *To_Do*.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 16/04/2015	**Fecha Fin:** 19/04/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se diseña una interfaz que permita a los usuarios crear los archivos *To_Do* con las características que propone la presente investigación.	

Tabla 20: Tarea de ingeniería. Implementar funcionalidad Crear archivo To_Do.

Tarea de Ingeniería	
Número Tarea: 14	**Número de Historia de Usuario:** 9
Nombre de Tarea: Implementar funcionalidad Crear archivo *To_Do*.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1 semana.
Fecha Inicio: 19/04/2015	**Fecha Fin:** 26/04/2015

Programador Responsable: Manuel Enrique Peiso Cruz.
Descripción: se implementar la funcionalidad “Crear archivo *To_Do*”, que permite crear el archivo propuesto con las características establecidas por la presente investigación.

Tabla 21: Tarea de ingeniería. Implementar funcionalidad Mostrar requisitos guardados.

Tarea de Ingeniería	
Número Tarea: 15	**Número de Historia de Usuario:** 10
Nombre de Tarea: Implementar funcionalidad Mostrar requisitos guardados.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0,5 semana.
Fecha Inicio: 26/04/2015	**Fecha Fin:** 29/04/2015
Programador Responsable: Manuel Enrique Peiso Cruz.	
Descripción: se implementar la funcionalidad “Mostrar requisitos guardados”, que permite mostrar los requisitos que se hayan detectado en revisiones previas y hayan sido guardados por el usuario en la base de datos.	

3.3 Estándar de codificación

Se sigue el estándar de programación sugerido por la arquitectura de Drupal.

- Cuando se escribe en PHP, siempre se deben utilizar las etiquetas <?php y ?>, y en ningún caso la versión corta <? y ?>.
- Cada sentencia condicional debe llevar sus respectivas llaves.
- En el fichero “.module” se omite la etiqueta de cierre de PHP (?>).
- En los archivos de plantilla .tpl.php, cada fragmento de PHP debe llevar sus correspondientes etiquetas de apertura y cierre, para diferenciarlo del código HTML.
- Se pueden usar tanto las comillas simples ('cadena') como las comillas dobles ("cadena") para delimitar las cadenas de caracteres.

También se usa para la implementación de la solución el estándar de código *CamelCase,* específicamente el tipo *UpperCamelCase,* que define que cada palabra que compone el nombre

de los métodos deben comenzar con mayúscula.

3.4 Pruebas

En el proceso de desarrollo de un *software* la realización de pruebas constituye una actividad importante para verificar la calidad del sistema implementado. Las mismas proporcionan distintos criterios para generar casos de prueba, los cuales se agrupan en métodos de caja blanca y caja negra. El método de caja negra realiza las pruebas sobre la interfaz de la aplicación, permitiendo encontrar errores de interfaz y de rendimiento principalmente. Debido al tipo de aplicación a desarrollar fue seleccionado este método, definiendo como tipos de pruebas, la de funcionalidad, realizada para detectar no conformidades del cliente y de fiabilidad y rendimiento, utilizadas en las pruebas de carga y estrés. Las pruebas son aplicadas en diferentes tipos de niveles: de aceptación y de sistema. La metodología SXP propone la realización de los Casos de Prueba de Aceptación (CPA), los cuales se realizan en conjunto con los usuarios finales antes del despliegue. Se definen, por tanto, tres iteraciones de pruebas para lograr una mayor organización en el trabajo, donde en la primera iteración se realizan los CPA, en la segunda las pruebas de carga y estrés y en la tercera iteración las de validación de indicadores y las funcionales (50).

3.4.1 Primera iteración: Casos de Pruebas de Aceptación

Tabla 22: Caso de prueba. Mostrar paquetes teniendo en cuenta sus clasificaciones.

Caso de prueba de aceptación	
Código Caso de Prueba: 01	**Nombre Historia de Usuario:** Mostrar paquetes teniendo en cuenta sus clasificaciones.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Mostrar paquetes”, mostrando datos correctamente.	

Condiciones de Ejecución: acceder a la aplicación “Metodología para el desarrollo de distribuciones GNU/Linux”.
Entrada / Pasos de ejecución: 1. Seleccionar la opción *ToDo*. 2. Seleccionar la opción Aplicación.
Resultado Esperado: debe aparecer un listado de clasificaciones en la región izquierda de la pantalla y un listado de checkbox en la región derecha con los paquetes que se encuentran almacenados en el repositorio. Al seleccionar alguna clasificación, debe aparecer en la parte derecha un listado de checkbox con los paquetes que pertenecen a dicha clasificación.
Evaluación de la Prueba: satisfactoria.

Tabla 23: Caso de prueba. Buscar requisitos de paquetes seleccionados.

Caso de prueba de aceptación	
Código Caso de Prueba: 02	**Nombre Historia de Usuario:** Buscar requisitos de paquetes seleccionados.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Buscar requisitos”, mostrando datos correctamente.	
Condiciones de Ejecución: deben existir paquetes en los cuales buscar.	
Entrada / Pasos de ejecución: 1. Seleccionar paquete. 2. Presionar el botón Buscar requisitos.	
Resultado Esperado: debe aparecer un área de texto con los nombres de los paquetes seleccionados y los requisitos que le faltan por implementar en caso de tener incorporado el archivo *To_Do*, o un texto que indica que el archivo no está presente.	
Evaluación de la Prueba: satisfactoria.	

Tabla 24: Caso de pruebas. Buscar paquetes.

Caso de prueba de aceptación

Código Caso de Prueba: 03	**Nombre Historia de Usuario:** Buscar paquetes.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Buscar paquetes”, mostrando datos correctamente.	
Condiciones de Ejecución: debe existir en la interfaz visual el campo de búsqueda.	
Entrada / Pasos de ejecución: 1. Insertar nombre de paquete en el campo de búsqueda. 2. Presionar el botón Buscar.	
Resultado Esperado: deben aparecer en la región izquierda las distintas clasificaciones de los paquetes y en la región derecha un listado de checkbox con el nombre de los paquetes que tengan alguna coincidencia con el nombre insertado.	
Evaluación de la Prueba: satisfactoria.	

Tabla 25: Caso de prueba. Guardar requisitos.

Caso de prueba de aceptación	
Código Caso de Prueba: 04	**Nombre Historia de Usuario:** Guardar requisitos.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Guardar requisitos”, almacenando la información correctamente.	
Condiciones de Ejecución: tener información acerca de los requisitos que le faltan por implementar a algunos paquetes en el área de resultados.	
Entrada / Pasos de ejecución: 1. Presionar el botón Guardar.	
Resultado Esperado: debe almacenarse en la base de datos la información acerca de los requisitos que le faltan por implementar a los paquetes revisados, en caso de que esa información no haya sido guardada anteriormente.	
Evaluación de la Prueba: satisfactoria.	

Tabla 26: Caso de prueba. Exportar requisitos.

Caso de prueba de aceptación	
Código Caso de Prueba: 05	**Nombre Historia de Usuario:** Exportar requisitos.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Exportar requisitos”, generando archivo externo con la información correctamente.	
Condiciones de Ejecución: tener información acerca de los requisitos que le faltan por implementar a algunos paquetes en el área de resultados.	
Entrada / Pasos de ejecución: 1. Presionar el botón Exportar.	
Resultado Esperado: debe generarse un archivo externo con la información de los requisitos que le faltan por implementar a los paquetes revisados, este archivo debe estar en formato PDF.	
Evaluación de la Prueba: satisfactoria.	

Tabla 27: Caso de prueba. Cancelar resultados.

Caso de prueba de aceptación	
Código Caso de Prueba: 06	**Nombre Historia de Usuario:** Cancelar resultados.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Cancelar resultados”, saliendo de la vista de resultados de la búsqueda.	
Condiciones de Ejecución: encontrarse en la interfaz que muestra los resultados de la búsqueda.	
Entrada / Pasos de ejecución: 1. Presionar el botón Cancelar.	
Resultado Esperado: debe volver a la interfaz que muestra los paquetes que se encuentran alojados en el repositorio.	
Evaluación de la Prueba: satisfactoria.	

Tabla 28: Caso de prueba. Actualizar base de datos.

Caso de prueba de aceptación

Código Caso de Prueba: 07	**Nombre Historia de Usuario:** Actualizar base de datos.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Actualizar base de datos”, guardando y mostrando datos correctamente.	
Condiciones de Ejecución: encontrarse en la interfaz principal del sistema.	
Entrada / Pasos de ejecución: 1. Presionar el botón Actualizar BD.	
Resultado Esperado: debe almacenar los paquetes en la base de datos con sus respectivas clasificaciones y mostrarlos en la vista principal del sistema.	
Evaluación de la Prueba: satisfactoria.	

Tabla 29: Caso de prueba. Crear archivo To_Do.

Caso de prueba de aceptación	
Código Caso de Prueba: 08	**Nombre Historia de Usuario:** Crear archivo *To_Do*.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Crear archivo *To_Do”*, creando el archivo en /tmp/to-do/ correctamente.	
Condiciones de Ejecución: encontrarse en la interfaz principal del complemento.	
Entrada / Pasos de ejecución: 1. Presionar el botón “Crear *To_Do*”. 2. Insertar los datos en los campos mostrados. 3. Presionar el botón “Crear”.	
Resultado Esperado: debe aparecer el archivo *To_Do* en /tmp/to-do/ con los datos insertados por el usuario.	
Evaluación de la Prueba: satisfactoria.	

Tabla 30: Caso de prueba. Mostrar requisitos guardados.

Caso de prueba de aceptación	
Código Caso de Prueba: 09	**Nombre Historia de Usuario:** Mostrar requisitos guardados.
Nombre de la persona que realiza la prueba: Manuel Enrique Peiso Cruz.	
Descripción de la Prueba: prueba a la funcionalidad “Mostrar requisitos guardados”, mostrando los datos correctamente.	
Condiciones de Ejecución: encontrarse en la interfaz principal del sistema.	
Entrada / Pasos de ejecución: 1. Presionar el botón “Mostrar revisados”.	
Resultado Esperado: debe mostrar los requisitos que se hayan detectado en búsquedas previas y que se almacenaron en la base de datos por el usuario.	
Evaluación de la Prueba: satisfactoria.	

3.4.2 Segunda iteración: pruebas de carga y estrés

Las pruebas de rendimiento fueron realizadas utilizando la herramienta Apache JMeter en su versión 2.8, diseñada para pruebas de carga de comportamientos funcionales y la medición del rendimiento. Prueba la resistencia y analiza el rendimiento en diferentes tipos de carga. El entorno en que fueron realizadas cumple con las siguientes características: 1 PC cliente, con un procesador Core i3 y 4Gb de RAM.

A continuación se presenta el plan de pruebas de rendimiento para algunas acciones que los usuarios pueden realizar al conectarse a la aplicación.

Para una muestra de 20 usuarios conectados concurrentemente, con un período de subida de 1 segundo (tiempo que espera cada usuario para realizar una petición) la aplicación generó los siguientes reportes:

Tabla 31: Resultado de pruebas de rendimiento.

Funcionalidad	URL	Min	Máx.	Kb/seg	Rendimiento	Avg.
Mostrar todos los paquetes.	drupal-7.26/ToDo/aplicaciones	1	1896	1636.9	20,0/seg.	46
Buscar requisitos que faltan por implementar.	drupal-7.26/ToDo/aplicaciones	1	3730	6026,9	39,7/seg.	129
Guardar requisitos.	drupal-7.26/ToDo/aplicaciones?pkg=respuesta	1	2434	6700.4	44.1/seg.	102
Exportar a archivo externo.	drupal-7.26/ToDo/aplicaciones?pkg=respuesta	5	13	14.4	6,0/seg.	7
Buscar paquetes.	drupal-7.26/ToDo/aplicaciones	1	1324	693.3	10,2/seg.	40
Mostrar paquetes de comunicación.	drupal-7.26/ToDo/aplicaciones&pkg=Comunicacion	1	905	805.6	15,4/seg.	22

Descripción de las pruebas

- La funcionalidad “Mostrar todos los paquetes” fue probada haciendo clic en el enlace al complemento *“ToDo”* que se encuentra en la interfaz principal de la aplicación “Metodología para el desarrollo de distribuciones GNU/Linux” y luego seleccionando la opción “Aplicación”.
- La funcionalidad “Buscar requisitos que faltan por implementar” fue probada seleccionando el paquete “pidgin” y pulsando el botón “Buscar requisitos”.
- La funcionalidad “Guardar requisitos” fue probada presionando el botón “Guardar” luego de haber detectado previamente los requisitos que le faltan por implementar al paquete “pidgin”.
- La funcionalidad “Exportar a archivo externo” fue probada presionando el botón “Exportar” luego de haber detectado previamente los requisitos que le faltan por implementar al paquete “pidgin”.

- La funcionalidad “Buscar paquetes” fue probada insertando el nombre “pidgin” en el campo de búsqueda y presionando el botón “Buscar”.
- La funcionalidad “Mostrar paquetes de comunicación” fue probada dando clic en la clasificación “Comunicación” de la vista principal del complemento.

Los datos de la tabla 31 muestra como resultado, que para las pruebas de rendimientos realizadas a las funcionalidades, los tiempos máximos de respuestas de la aplicación fueron de 3730 milisegundos, con un rendimiento de 22,5 segundos por cada 20 peticiones simultáneas. La interpretación de estos resultados demuestra que la aplicación se comporta satisfactoriamente, atendiendo a los tiempos de respuestas trazados por los desarrolladores, el cual debe oscilar, teniendo en cuenta la complejidad de las operaciones realizadas por el complemento, entre 10 y 50 segundos por cada 20 peticiones simultáneas realizadas.

3.4.3 Tercera iteración: comparación de indicadores

Indicador 1: agiliza el proceso de actualización de las aplicaciones.

En entrevista realizada al ingeniero Héctor Pérez Baranda, se determinó que en el DSO se establece un tiempo de una semana para identificar los requisitos que hay que implementarle a la actualización del sistema en el que se esté trabajando. El tiempo en que se logra realizar esta operación está en dependencia de la experiencia que se tenga en dicha aplicación y en el lenguaje de programación en que esté desarrollada, pero mayormente se utiliza toda la semana establecida para identificarlos.

En este período se hace imposible revisar todos los paquetes que intervienen en la actualización, así que se priorizan los de mayor importancia. Esto provoca que haya elementos que no se tengan en cuenta. El complemento desarrollado soluciona este problema, pues permite la revisión de todos los paquetes involucrados.

A continuación se muestra una tabla con el tiempo de respuesta del complemento a la hora de

identificar los requisitos que le faltan por implementar a algunos paquetes escogidos al azar y que poseen el archivo *To_Do*.

Tabla 32: Tiempos de respuesta de la aplicación.

Paquete	Tiempo de respuesta
pidgin	9,07 seg
amavisd-new	6,13 seg
gnokki	4,79 seg
mgetty	3,89 seg
fritzing	9,61 seg
owfs	3,04 seg
qmk-groundstation	3,05 seg

Luego de revisar, con la utilización del sistema, todos los paquetes del repositorio, se determinó que el tiempo que necesita el complemento para realizar la operación es de aproximadamente 5 horas y 44 minutos.

Si se interpreta lo anteriormente expuesto se puede afirmar que el uso del complemento *ToDo* agiliza el tiempo de actualización de las aplicaciones, si se tiene en cuenta que el proceso de identificación de requisitos es mucho más rápido.

Indicador 2: se tiene mayor control del proceso de identificación de requisitos

El complemento desarrollado posee una ventaja de vital importancia en el proceso de identificación de requisitos: es capaz de establecer la trazabilidad entre los requisitos identificados y el código, pues dichos requisitos pertenecen a un paquete determinado donde se enmarca su desarrollo. Todo esto facilita el proceso de implementación de la funcionalidad y permite poseer mayor control del proceso en general, además de conocer hasta dónde están desarrollados los paquetes.

3.4.4 Resultados de las pruebas a las funcionalidades

Se probaron todos los casos de pruebas que responden a las funcionalidades del complemento desarrollado. En total fueron detectadas 23 no conformidades, las cuales fueron resueltas, donde los principales errores detectados se relacionan con el diseño, validaciones incorrectas, ortografía y rendimiento.

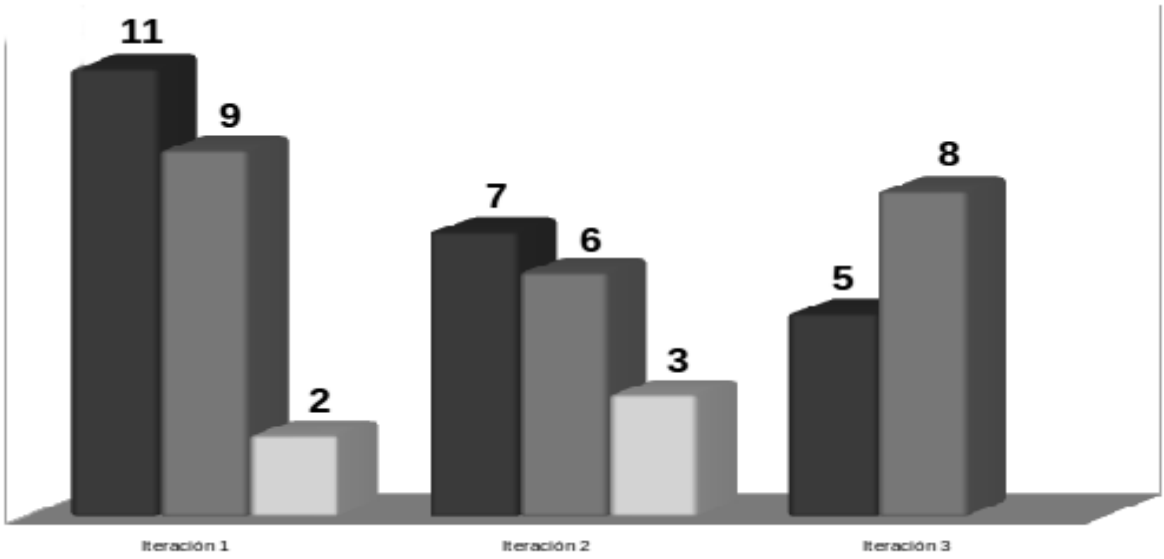

Figura 7: Iteraciones de implementación.

En la Figura 16 se muestra una gráfica donde se desglosan las no conformidades detectadas en las tres iteraciones realizadas. En la primera iteración se obtuvieron 11 no conformidades, de las cuales se resolvieron 9 y las 2 restantes quedaron pendientes para la segunda iteración, en donde se detectaron 7 no conformidades y fueron resueltas 6, que se descomponen en las 2 pendientes y 4 detectadas en la presente iteración, de igual manera quedaron 3 no conformidades por resolver. En la tercera iteración se detectaron 5 no conformidades las cuales

fueron resueltas junto con las que quedaron de la iteración precedente.

3.5 Conclusiones del capítulo

- El establecimiento de la prioridad de los requisitos funcionales permitió la realización del plan de iteraciones, donde se determinaron qué requisitos fueron implementados en cada iteración.
- La necesidad de implementar el sistema conllevó a la realización de un conjunto de tareas de ingeniería que permitieron trazar los pasos a seguir a la hora de realizar la aplicación.
- La obtención del prototipo funcional de la aplicación implicó la revisión de cada uno de los elementos que la componen para encontrar no conformidades a través del diseño de los casos de pruebas, los cuales fueron ejecutados con satisfacción sobre los requisitos funcionales implementados.
- Teniendo en cuenta que el complemento desarrollado fue realizado utilizando Drupal se determinó la utilización del estándar de codificación que usa dicho CMS así como el *CamelCase*.
- Se obtuvo como resultado práctico un complemento que permite la identificación de los requisitos que le faltan por implementar a las aplicaciones que se encuentran alojadas en un repositorio de código fuente.
- A través de la comparación de los indicadores definidos en la idea a defender antes y después del desarrollo del complemento, se concluyó que lo planteado por la misma se cumple.

Conclusiones generales

Al término de la presente investigación se evidencia el cumplimiento de los objetivos planteados al inicio de la misma, lo que permitió arribar a las siguientes conclusiones:

1. El estudio de las metodologías de desarrollo de *software*, la manera en que se realiza la identificación de requisitos en CESOL, así como la estructura y actividades de un repositorio de código fuente y binario, permitió determinar que la incorporación de un archivo llamado *To_Do* con los requisitos que le faltan por implementar a las aplicaciones, en las carpetas debian, de los paquetes fuente del repositorio, es una forma certera de almacenar dicha información.
2. El análisis y diseño de la propuesta de solución permitió obtener un sistema completamente libre, funcional para los repositorios de códigos fuente, que automatiza el proceso de identificación de requisitos a la hora de realizarle actualizaciones a las aplicaciones que se encuentran alojadas en los mismos.
3. El diseño y ejecución de pruebas de aceptación, divididas en pruebas de caja negra, carga y estrés y cumplimiento de indicadores permitió obtener un complemento con calidad, que cumple puntualmente con todos los requisitos planteados.

Recomendaciones

Existen varias funcionalidades que le aportarían mayor alcance a la presente investigación, pues aumentaría el valor de la misma. Algunas de ellas se recomiendan a continuación:

- Lograr incorporar el archivo *To_Do* a todas las aplicaciones que se desarrollan en CESOL.
- Lograr que los reportes generados por la aplicación Auditoría de Código Fuente (ACF), que se dedica a la detección de errores en el código de las aplicaciones que están almacenadas en el repositorio, se incorporen a la información del archivo *To_Do* como requisitos que faltan por implementar.

Bibliografía

1. SÁNCHEZ, Isabel Bárbara. Aplicación de soporte a la Metodología de desarrollo de distribuciones GNU/Linux, Nova-OpenUp. La Habana: Centro de Software Libre y Código Abierto, Universidad de las Ciencias Informáticas, 2014.

2. FERNÁNDEZ, Yusleydi. Metodología para desarrollar la distribución cubana de GNU/Linux Nova. La Habana: Centro de Software Libre y Código Abierto, Universidad de las Ciencias Informáticas, 2013.

3. PRESSMAN, Roger S. Ingeniería de software; un enfoque práctico. 6. McGrau-Hill, 2005.

4. BLANCO, Carlos. Patrones de diseño. Ingeniería de software 1. Universidad de Cantabria.

5. Definiciones de Principales Términos más Usados en Internet. [online]. [Accessed 8 February 2015]. Available from: http://www.informaticamilenium.com.mx/es/temas.html

6. Complemento (informática) - Wikipedia, la enciclopedia libre. [online]. [Accessed 8 February 2015]. Available from: http://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)

7. ¿Cuál es la definición de Plugin? [online]. [Accessed 8 February 2015]. Available from: http://www.alegsa.com.ar/Dic/plugin.php

8. ACHILLO, Edgar Pedro Garcia. Función y Procedimiento. [online]. 2013. Available from: http://www.cienciaytecnologia.gob.bo/vcyt2012/uploads/cap-2-funciones-y-procedimientos.pdf.

9. ¿Cuál es la definición de módulo (programación)? [online]. [no date]. Available from: http://www.alegsa.com.ar/Dic/modulo.php

10. MCGRAW-HILL. Sistemas Gestores de Contenidos [online]. McGraw-Hill, 2009. Available from: http://bit.ly/1ctYa4C

11. Drupal. Wikipedia, la enciclopedia libre [online]. 2015. [Accessed 8 February 2015]. Available from: http://es.wikipedia.org/w/index.php?title=Drupal&oldid=79224670

12. Drupal - EcuRed. [online]. [Accessed 8 February 2015]. Available from: http://www.ecured.cu/index.php/Drupal#Caracter.C3.ADsticas_generales.C2.A0

13. Introducción al HTML - Virtualnauta Tutorial HTML. [online]. [Accessed 9 February 2015]. Available from: http://www.virtualnauta.com/html-introduccion

14. BAKKEN, SÆTHER, SCHMID, STIG and EGON. Manual de Php. PHP Documentation Group. [online]. 2003. Available from: http://www.php.net/docs.php.

15. Guía Breve de CSS. [online]. [Accessed 9 February 2015]. Available from: http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo

16. JavaScript | MDN. [online]. [Accessed 9 February 2015]. Available from: https://developer.mozilla.org/es/docs/Web/JavaScript

17. Python, ¿qué es? [online]. [Accessed 10 February 2015]. Available from: http://es.wikipedia.org/wiki/Python

18. ¿Qué es un servidor web ? [online]. [Accessed 10 February 2015]. Available from: http://www.misrespuestas.com/que-es-un-servidor-web.html

19. MÁRQUEZ, J., Sampedro, L. and VARGAS, F. Instalación y configuración de Apache, un servidor Web gratis. Red de Revista Científica de América Latina y el Caribe [online]. España y Portugal, 2002. Available from: http://redalyc.uaemex.mx/pdf/852/85201202.pdf

20. ¿Cuál es la definición de IIS (Internet Information Services)? [online]. [Accessed 10 February 2015]. Available from: http://www.alegsa.com.ar/Dic/iis.php

21. ESPINOSA, Humberto. PostgreSQL Una Alternativa de DBMS Open Source [online]. 2005. Available from: http://bit.ly/1PFm8qw

22. TOLEDO, Enrique Alma, AYALA, Maldonado Jesús, ORTEGA, Yunko Nakamura and TOLEDO, Nogueron Noretti. ¿Qué es MySQL? [online]. 2009. Available from: http://bit.ly/1PFm9ur

23. CIDI. CIDI, Centro de Ideoinformática. Base Tecnológica. Universidad de las Ciencias Informáticas. Ciudad de La Habana, 2012.

24. Sublime Text, un sofisticado editor de código multiplataforma. [online]. [Accessed 8 February 2015]. Available from: http://www.genbeta.com/herramientas/sublime-text-un-sofisticado-editor-de-codigo-multiplataforma.

25. Visual Paradigm for UML - Libere las revisiones de la transferencia directa y del software | CNET Download.com. [online]. [Accessed 10 February 2015]. Available from: http://descargar.cnet.com/Visual-Paradigm-for-UML/3000-2247_4-42700.html

26. BARZANALLANA, Rafael. Apuntes. Ingeniería del software. Sistemas Informáticos. Nivel de madurez sofware. Informática Aplicada a la Gestión Pública. [online]. 2005. [Accessed 12 February 2015]. Available from: http://www.um.es/docencia/barzana/IAGP/Iagp2.html

27. PEÑALVER, G., MENESES, A. and GARCÍA, S. SXP, METODOLOGÍA ÁGIL PARA EL DESARROLLO DE SOFTWARE. Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba, 2010.

28. SOMMERVILLE, Ian. Ingeniería de software. 8. China Machin Press, 2006. ISBN 7-111-19770-4.

29. SARAIVA, Edson de Almeida. Requisitos de software. Universidad de San Judas Tadeo. 2014.

30. LOAIZA, Vanesa Carolina Carvajal and Zorro, Laura Catalina Jiménez. Análisis de herramientas de administración de requerimientos: open source requirement management tool – OSRMT. Pontificia universidad Javeriana, 2010.

31. MOLINA, J. C. and TORRES, M. C. Análisis de requerimientos usando BPMN. . 2010. Vol. 11, p. 85–97. Revista Colombiana de Computación

32. ARIAS, Rodríguez and GARCÍA. Guía para la trazabilidad de requisitos en el Visual Paradigm 8.0. Universidad de las Ciencias Informáticas, La Habana, Cuba, 2010.

33. UCI. Libro de proceso para la administración de requisitos. CMMI. Universidad de las Ciencias Informáticas, La Habana, Cuba, 2014.

34. Redefinición de los procesos de desarrollo producción en la Universidad de las Ciencias Informáticas. [online]. [Accessed 5 February 2015]. Available from: http://mejoras.prod.uci.cu/

35. LETELIER, P. Rational Unified Process [online]. Universidad Politécnica de Valencia, Valencia, España, 2005. Available from: https://pid.dsic.upv.es

36. JACOBSON, Ibar, BOOCH, Grady and RUMBAUGH, James. El proceso unificado de desarrollo de software. Madrid : Addison-Wesley, 2000. ISBN 0-201-57169-2.

37. WAKE, W.C. Extreme Programming Explored. Addison-Wesley, 2002.

38. JEFFRIES, R., ANDERSON, A. and HENDRICKSON, C. Extreme Programming Installed. Addison-Wesley, 2001.

39. NEWKIRK, J. and MARTIN, R.C. Extreme Programming in Practice. Addison-Wesley, 2001.

40. BECK, K. Extreme Programming Explained. Embrace Change. Pearson Education, 1999.

41. PALACIO, Juan. El modelo scrum. Navegapolis, 2006.

42. SUTHERLAND, Jeff and SCHWABER, Ken. The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework. Open view labs, 2010.

43. CALABRIA, Luis. Metodología FDD. Cátedra de Ingeniería de software de la Universidad ORT de Uruguay, 2003.

44. ¿Que son los repositorios en Linux? Útiles, rápidos y seguros | OvToaster. [online]. [Accessed 8 February 2015]. Available from: http://ovtoaster.com/repositorios-linux/

45. TORRES, José María Corral. Debian, tema 9. Todo lo relacionado con los repositorios de Debian. 2009.

46. ISO. Organización internacional de estandarización (iso). 2012.

47. TORRELAGUNA. Introducción a Linux. Administración básica del sistema. 2010. Ministerio de Educación.

48. DE GIUSTI, Marisa. Bibliotecas y Repositorios Digitales Tecnología y Aplicaciones: la preservación en el RI. . Universidad Nacional de la Plata. 2006.

49. APPLETON, B. Patterns and Software: Escencial Concepts and Terminology [online]. [no date]. Available from: http://www.enteract.com/~bradapp/docs/patterns-intro.html

50. Ingeniería de Software 2. In : Introducción a la Verificación y Validación de software. Técnicas estáticas y dinámicas de verificación y validación. Plan y estrategia de pruebas.