# Universidad de las Ciencias Informáticas

## Facultad 1


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

## Título:

Herramienta de análisis del repositorio de Nova.

## Autor:

Daniel Rolando Escarret de Lázaro

## Tutores:

Mtr. Yoandy Pérez Villazón.  Ing. Jesús Rabelo Pérez.

Ciudad de La Habana

2016

## Pensamiento

“Tu trabajo va a llenar gran parte de tu vida, la única manera de estar realmente satisfecho es hacer lo que creas, es un gran trabajo y la única manera de hacerlo es amar lo que haces. Si no lo has encontrado aún, sigue buscando. Como con todo lo que tiene que ver con el corazón, sabrás cuando lo hayas encontrado.”

Steve Jobs

# Declaración de autoría

Declaro ser el único autor de este trabajo y concedo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma con carácter exclusivo. Para que así conste firmo la presente a los ___ días del mes de _____ del año 2016.

_____

Daniel Rolando Escarret de Lázaro

_____ _____

Mtr. Yoandy Pérez Villazón. Ing. Jesús Rabelo Pérez.

# Dedicatoria

A mi madre, hermanos y abuela. A la familia en general.

A mis compañeros y amigos de la UCI.

# Agradecimientos

*A mi madre Nelys, por ser el epicentro de mi educación.*

*A mis hermanos Danilo y Dianelys, por ser quienes me apoyaron en todo momento.*

*A mi abuela Marbelia, por transmitirme sus conocimientos y enseñanzas las cuales me servirán para toda la vida.*

*Al resto de la familia, especialmente a mi ahijado Kelvin.*

*A mis compañeros y hermanos del 1501 y el 1502, quienes estuvieron a mi lado en las buenas y en las malas, especialmente al grupo de la Baticueva y sus pupilos: Ariel, Pita, Junquera, Yobal, Joseph, Marbier y Enmanuel así como a Marlon, Emilio, Ricard, Jesúz, Dainel, Liorge y Eylin.*

*Al departamento de Nova, especialmente a Dayrelys y a Manuel, gracias por existir.*

*A Kari por estar siempre ahí para mí.*

*A Blizzard, Square-Enix, Ubisoft, From Software y Konami por brindarme excelentes momentos para mí. Sigan así.*

*A todos los profesores que contribuyeron en mi formación profesional: este es su resultado.*

*También agradezco a los tutores por la inmensa ayuda brindada, sin ustedes este sueño no sería posible.*

*A todos los que de una forma u otra contribuyeron en mi formación profesional así como en el desarrollo de esta investigación. Mis bendiciones para ellos.*

# Resumen

Desde hace varios años Cuba está inmersa en una migración al uso del software libre, una tarea que se está llevando a cabo hoy día, lográndose ver en varias instituciones del país. GNU/Linux Nova es una de las distribuciones libres la cual cuenta con un repositorio de aplicaciones capaz de suplir las necesidades de informatización de los Órganos y Organismos de la Administración Central del Estado (OACE). En la actualidad los usuarios hacen rechazo a la distribución por múltiples inconsistencias en el repositorio.

El objetivo de la presente investigación es desarrollar una aplicación web que garantice la detección de inconsistencias sobre el repositorio de GNU/Linux Nova. Se documenta la investigación realizada con el fin de demostrar que, dada la situación problemática actual, se requiere la creación de un nuevo software que responda a las necesidades encontradas. También se expone un estudio comparativo sobre las tecnologías y tendencias de los sistemas de gestión de paquetes de software, seleccionando las más adecuadas como guía para la elaboración de la solución propuesta. La metodología de desarrollo aplicada fue AUP-UCI, recogiéndose en el presente documento los principales artefactos generados en cada una de sus iteraciones. El resultado final es una aplicación web que detecta las inconsistencias existentes en el repositorio de aplicaciones de GNU/Linux Nova, e informa además a los mantenedores del repositorio sobre los problemas detectados, generando informes asociados al proceso.

Palabras claves:

Aplicación web, dependencia, detectar, inconsistencia, repositorio.

# Índice de contenido

# Índice de figuras

# Índice de tablas

# Introducción

La Universidad de las Ciencias Informáticas (UCI) es una de las instituciones encargadas del proceso de soberanía tecnológica que actualmente se está desplegando en nuestro país, donde una de sus ramas más especializadas está dedicada a la realización e investigación de aplicaciones informáticas para satisfacer las necesidades de la migración a plataformas de código abierto, el centro de producción encargado de esta tarea es el Centro de Soluciones Libres (CESOL).

Entre los principales productos sobresalientes de la línea de Investigación y desarrollo a las que se dedica este centro se puede mencionar Nova, la cual es una distribución de GNU/Linux que brinda una gama de productos y servicios orientados a usuarios nacionales inexpertos en el área de las tecnologías de software libre y que estén experimentado un proceso de migración a las mismas. Esta distribución, con 10 años de creada, dispone de un repositorio de aplicaciones con más de 15 000 paquetes de software a los cuales los usuarios pueden acceder. A continuación, se muestra una tabla la cual refleja los datos acerca del despliegue de esta distribución.

*Tabla 1: Despliegue de la distribución GNU/Linux Nova*

| Instituciones | Computadoras de escritorio | Servidores |
|---|---|---|
| UCI | 4109 | |
| Red de soporte del sistema electoral cubano | 206 | |
| Fiscalía General de la República | | 203 |
| CECAM | 80 | 3 |

El despliegue en diversas instituciones del sistema operativo y la instalación por cientos de usuarios a lo largo de los años se ha visto afectada en ocasiones por problemas de inconsistencia existentes en el repositorio, cuestión que provoca rechazo en los usuarios hacia la distribución. Según algunos datos arrojados por una encuesta realizada (Ver Anexo#1), las personas encuentran problemas, un estudio más profundo del tema permitió conocer que esto se debe generalmente por dependencias insatisfechas en los paquetes o diferencias entre la información disponible en el fichero *Package* y los paquetes realmente disponibles en el repositorio. La única forma actual de detectar estas deficiencias está dada por la instalación de cada paquete de software en una computadora, cuestión compleja de verificar y que tarda una excesiva cantidad de tiempo.

Otra forma de detectar algunos de estos errores es trabajando con herramientas que solo realizan un trabajo parcial al esperado y que están orientadas a otra actividad, haciendo que el procesamiento se torne engorroso y en algunos casos difícil de ejecutar, especialmente porque algunas de estas aplicaciones trabajan sin una interfaz visual provocando el rechazo en algunos de los usuarios de Nova.

A partir de todo lo planteado anteriormente, se identificó el siguiente **problema de la investigación**: ¿Cómo contribuir a la detección de inconsistencias del repositorio de paquetes de la distribución GNU/Linux Nova?

Una vez identificado el problema se establece como **objetivo general**: Desarrollar una aplicación web que permita detectar inconsistencias en el repositorio de paquetes de la distribución GNU/Linux Nova e informar a los mantenedores sobre los problemas detectados.

Partiendo del objetivo general, se plantearon los siguientes **objetivos específicos**:

1. Caracterizar las soluciones de Código Abierto existentes que permitan medir la calidad y el grado de funcionalidad de un repositorio de software.
2. Diseñar una aplicación web que permita detectar inconsistencias de GNU/Linux Nova y generar un reporte para los mantenedores de paquetes con la información de cada paquete que posea alguna inconsistencia.
3. Implementar la solución propuesta.
4. Evaluar el resultado con la aplicación práctica de la solución sobre el repositorio de GNU/Linux Nova.

El **objeto de estudio** se centra en los repositorios de Linux basados en Debian mientras que el **campo de acción** se enmarca en el repositorio de GNU/Linux Nova.

**Idea a defender:**

Con el desarrollo de una herramienta que permita detectar inconsistencias en repositorios de paquetes de la distribución GNU/Linux Nova se contribuirá a un correcto mantenimiento de estos en la distribución GNU/Linux Nova.

Para dar cumplimiento a los objetivos específicos se planifican las siguientes **tareas de investigación**:

1. Definición de los conceptos asociados a los repositorios de software para una distribución GNU/Linux.
2. Comparación de los sistemas homólogos encontrados.
3. Definición de los requisitos funcionales y no funcionales de la solución propuesta.
4. Descripción de la arquitectura propuesta.

5. Codificación de las funcionalidades.
6. Diseño de los casos de prueba.
7. Descripción de los resultados de las pruebas a partir de la aplicación práctica teniendo como caso de estudio el repositorio de GNU/Linux Nova.

En el desarrollo de la investigación se utilizan los siguientes métodos científicos:

**Métodos teóricos:**

- **Analítico-Sintético:** Posibilitó el análisis de elementos particulares relacionados con los sistemas de gestión de inconsistencias, a partir de su estudio permitió definir las características fundamentales para una correcta ejecución de la herramienta.
- **Inductivo-Deductivo:** Permitió integrar los métodos y funcionalidades de diferentes herramientas para la gestión de inconsistencias, así como sus patrones de diseño para resolver problemas particulares de la solución en desarrollo.

**Métodos empíricos:**

- **Entrevista**: Método utilizado para la obtención de información.

El documento se encuentra estructurado en 3 capítulos, la fundamentación teórica, el análisis y diseño de la solución propuesta y la implementación y pruebas al sistema. Las conclusiones generales, la bibliografía general utilizada, el glosario de términos y los anexos sirven de sustento para lograr la total comprensión de la investigación. La estructura de los capítulos se define a continuación: **Capítulo 1:** Fundamentación teórica.

En este capítulo se hace referencia a todos los elementos teóricos a tener en cuenta en la investigación del sistema a implementar, haciendo énfasis en los principales sistemas de gestión de inconsistencias en los repositorios existentes. Además, se analizan mecanismos, técnicas, herramientas y lenguajes para dar solución a la problemática propuesta.

**Capítulo 2:** Análisis y diseño de la solución propuesta.

Se hace alusión a la solución propuesta, planteándose la forma en la que se va a desarrollar la misma, así como el trabajo con las herramientas que se utilizarán para su implementación. Además, se describe el proceso ágil basado en la metodología AUP-UCI.

**Capítulo 3:** Implementación y pruebas.

Este capítulo describe las tareas relacionadas con la fase de implementación y pruebas de las funcionalidades del sistema. Se completa el desarrollo del sistema basado en la arquitectura definida.

# Capítulo 1: Fundamentación teórica

## 1.1 Introducción

La distribución GNU/Linux Nova es uno de los principales pilares en el proceso de migración de software libre en el país. Al estar equipada con múltiples servicios y con repositorio bien estructurado es capaz de satisfacer las necesidades de todos los usuarios, lo que la convierte en una opción viable para los mismos; por eso el soporte del repositorio es un proceso que se está haciendo cada día más popular y a la vez necesario. En el presente capítulo se tratarán algunos de los aspectos más significativos asociados al problema de la investigación planteado y que son necesarios para su desarrollo, además de hacer un estudio de algunas aplicaciones de software libre relacionadas con el tema. Por último, se define la metodología de investigación, así como las herramientas y tecnologías a utilizar durante la implementación de la solución propuesta.

## 1.2 Conceptos fundamentales

Los conceptos aquí presentados son el resultado de la investigación y revisión cuidadosa de la bibliografía disponible a nivel internacional, nacional y en el ámbito de la UCI. Ofrecen una panorámica general sobre el dominio del problema:

### 1.2.1 Paquete

Es una colección de archivos de código fuente o binarios los cuales tienen un conjunto de archivos de instrucciones que especifican qué hacer con cada uno de ellos. Todos los archivos van comprimidos

según un formato especial que depende de la distribución [2]. Los paquetes se pueden diferenciar de la siguiente manera:

Paquetes de fuentes (comprimidos, pero no empaquetados): se reconocen por la extensión: .tgz

- **TGZ:** Es un archivo de paquetes específico para Unix, comprimido con el compresor Gnu Zip. Es un paquete de código fuente, ocupado para contener aplicaciones, y su código fuente, para no tener que crear un tipo de paquete específico para cada distribución. A diferencia de los paquetes .deb, o .rpm, este no contiene instrucciones particulares de instalación para cada distribución, por lo que la instalación del contenido deberá ser compilado por el usuario.

Paquetes binarios: contienen la información necesaria para reconstruir una aplicación en un sistema nuevo, sin necesidad de encontrarse en la misma computadora; los más comunes son:

- **DEB:** Contienen ejecutables, archivos de configuración, páginas de información, derechos de *copyright* y otras documentaciones, los paquetes Debian se colocan en archivos .deb. Una desventaja de este tipo de paquetes, es su sistema de actualización, debido a que, se necesita tener todos los archivos, como si se tratase de una nueva instalación. Estos paquetes también son usados por distribuciones basadas en la distribución Debian, algunas de estas, son: Ubuntu, Kubuntu, Linux Mint, entre otras.
- **RPM:** Por sus siglas en inglés *Redhat Package Manager*, este tipo de paquete para Linux fue desarrollado para la distribución de Red Hat, con el fin de crear un sistema fácil de crear e instalar. Actualmente todas las distribuciones basadas en Red Hat emplean los paquetes RPM, algunas de ellas son: Fedora y openSuse. Una poderosa ventaja, de este tipo de paquetes sobre otros, es su forma de actualización para las aplicaciones, estos, no necesitan tener los mismos datos que el

instalador original, solamente puede incluir (si se desea) los archivos que se actualizarán, esto reduce el peso del paquete.

Paquetes de fuentes empaquetadas: Típicamente .ebuild.

- **Ebuild:** Un ebuild es un script construido en el lenguaje de programación Bash, el cual cuenta en su primera parte declaraciones de variables, en una segunda parte cuenta con todas las funciones que tiene el ebuild. En los ebuilds se puede definir funciones que controlen el proceso de compilación e instalación de paquetes.

### 1.2.2 Repositorio de paquetes

Los sistemas de la familia GNU/Linux usan, por regla general, la gestión de paquetes para controlar las aplicaciones y bibliotecas que se instalan y/o desinstalan en el sistema. Estos sistemas de gestión obtienen los paquetes desde alguna ubicación específica que puede encontrarse disponible a través de la red o en algún dispositivo físico como discos duros, compactos o cualquier dispositivo de almacenamiento externo. A estas ubicaciones se les conoce como repositorios de paquetes de software. Existen distintas configuraciones estructurales para estos repositorios, en dependencia de qué sistema de gestión se utilice, pero todas coinciden en que en ellos se pueden encontrar los paquetes de software disponibles para instalar en la distribución que los mantenga. Otro detalle digno de hacer notar reside en que los paquetes incluidos en los repositorios pueden ser de código fuente o pre-compilados.

A diferencia de los ordenadores personales o de escritorio, los repositorios suelen contar con sistemas de respaldo (*backup*) y mantenimiento preventivo y correctivo, lo que hace que la información se pueda recuperar en el caso que la máquina quede inutilizable [1].

En el caso particular del repositorio de Nova la estructura implementada posee la siguiente descripción:

- **Principal**: El componente principal, contiene los paquetes fundamentales compatibles con todas las versiones de Nova además de las aplicaciones propuestas por la comunidad.
- **Extendido**: Contiene aquellos paquetes que quedaron heredados de otras distribuciones que son compatibles con Nova.

### 1.2.3 Dependencia

Es un paquete requerido por un instalador de paquetes para hacer funcionar correctamente un programa después de instalarlo. Generalmente son bibliotecas de código o programas accesorios. Un programa puede tener múltiples dependencias, las cuales no siempre vienen instaladas en la distribución Linux elegida por diversas razones, por no ser necesarias, por aumentar demasiado el tamaño del archivo ISO, entre otros. No todas las dependencias son absolutamente necesarias, algunas sólo dan soporte adicional para ciertas funciones o efectos visuales que extienden las características del programa en sí mismo [3].

Las dependencias se dividen en dos categorías:

- **Dependencias de compilación:** Las dependencias de compilación son aquellas que se necesitan que estén instaladas en el sistema para que el programa a instalar pueda realizar la operación de compilar.
- **Dependencias de ejecución:** Son las que el sistema necesita para que se pueda ejecutar.

Un programa puede tener ciertas dependencias de compilación y otras de ejecución. Sin embargo, puede suceder que alguna de las dependencias que tenga el paquete de compilación sean las mismas que las de ejecución.

### 1.2.4 Inconsistencia

Inconsistencia es el antónimo de consistencia que significa fuerza y rigidez de alguna situación o de alguna cosa [4]. Por lo que inconsistencia refiere al efecto contrario, a aquello que es vulnerable a lo que pueda alterar su funcionamiento, aquello que no es firme ni sólido. En el ámbito del software libre una inconsistencia describe a una dependencia insatisfecha de un paquete, el cual se puede encontrar en un repositorio. Dentro de las inconsistencias más comunes están:

- **Paquete incompleto:** Esto ocurre cuando un paquete no se puede ejecutar o instalar debido a que se encuentra dañado o fue subido incompleto.
- **Dependencias incumplidas:** Se detecta cuando las dependencias de un paquete no se encuentran.
- **Inconsistencia de versión:** El caso más común es cuando un paquete dependa de otro y el mismo no se encuentre en la versión especificada. Esta inconsistencia es muy común cuando se actualiza el repositorio.

## 1.3 Análisis de herramientas para la detección y corrección de inconsistencias en repositorios de Linux

A continuación se realiza el estudio de las herramientas encargadas de la detección y corrección de inconsistencia en repositorios, dándole un enfoque a estas soluciones basadas en la distribución GNU/Linux Nova, aunque se tiene en cuenta en todo momento los aportes de las aplicaciones, tanto de requisitos funcionales como no funcionales. Se enuncian una serie de características y desventajas de las mismas, permitiendo arribar a conclusiones que consoliden el proceso de desarrollo de la herramienta.

### 1.3.1 Summon

Nombrado Summon por el Proyecto Nova surge por la necesidad de eliminar la gestión de paquetes a nivel de código fuente. La idea inicial fue escribir la herramienta desde cero utilizando la plataforma Mono, pero una investigación profunda arrojó como resultado, que era más factible la utilización de alguno de los gestores de paquetes existentes compatibles con la base Gentoo. Se valoraron las opciones Ututo-get y Entropy, pertenecientes al Proyecto Ututo y Sabayon Linux respectivamente, eligiéndose la última por ser la opción más adecuada tecnológicamente, así como por estar programada en Python, un lenguaje ya conocido en el proyecto.

La herramienta permite la gestión eficiente de los paquetes además de garantizar una mayor eficiencia y control sobre los procesos de instalación y desinstalación y actualización de software en el sistema operativo NOVA. Posee una interfaz gráfica sencilla que se integra sin inconveniente alguno con Gnome, entorno predefinido de la distribución.

Sin embargo, la asimilación de Entropy generó un nuevo problema. Al existir los paquetes ya compilados en el repositorio se limitaba la libertad del usuario de escoger si deseaba instalar el software con (o sin) alguna funcionalidad en específico; sino que debía instalar las funcionalidades que se le habilitaban al paquete al compilarlo antes de incluirlo al repositorio [5]. En la actualidad Summon está descontinuada.

### 1.3.2 Reprepro

Reprepro es una herramienta para el manejo local de repositorios de paquetes de Debian. Presenta la capacidad de almacenar ficheros ya sea que se incluyan manualmente o descargados de algún repositorio de software. Los almacena en un directorio /pool ordenado alfabéticamente en un árbol concebido para

este fin. Gestiona los ficheros de configuración y los paquetes de software a través de una base de datos, según se compile Reprepro por lo que no se necesita un gestor de base de datos convencional.

Es compatible con varios algoritmos para la encriptación de datos como sha1, sha256 y md5. Además, se integra a otras aplicaciones necesarias para la gestión de repositorio. Emplea llaves generadas por GnuPG para hacer posible la autenticación de los repositorios y los paquetes. Es compatible con los protocolos de transferencia de datos HTTP[1], FTP[2]. También se puede configurar para que funcione mediante un proxy.

Realiza la gestión de repositorio no limitándose solo a la descarga de los paquetes y creación de la estructura del repositorio, sino que es capaz de realizar la administración y manejo de los índices de los repositorios ubicados de forma local en el disco duro. Una vez creado un repositorio hace posible su gestión de forma variada y dinámica. Reprepro cuenta con un número de ficheros que hacen posible su fácil configuración tornándose en una versátil y potente herramienta. Permite la existencia de varias versiones del mismo paquete de software en la misma base de datos, esto sólo si se desea. Una característica que puede ser explotada es el contar con comandos cortos si se compara con otras herramientas homólogas, simplificando la progresiva implementación de la solución al abstraer al desarrollador de las especificidades de la herramienta [6].

Como dificultad con que cuenta el software en cuestión, se tiene que las actividades para la detección de inconsistencias forman parte de procesos secundarios dentro de la gama de utilidades, además de que la

[1]HTTP: O Hypertext Transfer Protocol (en español *protocolo de transferencia de hipertexto*) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

[2]FTP: O *File Transfer Protocol*, (en español *protocolo de transferencia de archivos*) es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en la arquitectura cliente-servidor.

base de datos con que este realiza sus operaciones es propensa a corromperse cuando el manejo de los repositorios no es el correcto.

### 1.3.3 APT

Es un sistema de gestión de paquetes creado por el proyecto Debian. APT simplifica en gran medida la instalación, actualización y eliminación de paquetes en los sistemas GNU/Linux, así como la actualización del índice de los paquetes disponibles en los repositorios. Entre sus principales características se encuentra el manejo automático de conflictos, actualización de archivos de configuración para las aplicaciones que así lo requieran y facilidad de uso (esto último sólo es válido para usuarios que acostumbran usar la terminal). No existe un programa APT en sí mismo, sino que es una biblioteca de funciones C++ que se emplea por varios programas de línea de comandos para distribuir paquetes. A continuación se explicarán dos funciones relacionados con la gestión de paquetes:

#### Apt-cache

Es una interfaz de líneas de comandos, la misma trabaja directamente sobre la APT permitiendo realizar una serie de operaciones sobre la caché de paquetes. “Apt-cache**”** no modifica el estado del sistema, pero proporciona operaciones de búsqueda en la información de los paquetes, de las cuales se puede obtener información muy útil.

Entre las principales opciones que ofrece se encuentran las siguientes:

- **add** fichero(s): Añade el nombre del paquete a los ficheros de índices de la caché de paquetes. Sólo para depuración.
- **showpkg** paquete(s): Muestra información acerca de los paquetes listados en la línea de órdenes. Los argumentos restantes se consideran nombres de paquetes. Por cada paquete se mostrarán

las versiones disponibles y los paquetes que dependen de él, así como los paquetes de que depende.

- **showsrc** paquete(s): Muestra todos los campos de los paquetes fuente que coinciden con los nombres de los paquetes suministrados. Se muestran todas las versiones, así como los paquetes que son binarios.
- **policy** [pkg…]: Está pensado para ayudar a depurar asuntos relacionados con el fichero de preferencias. Sin argumentos mostrará las prioridades de cada fuente. De forma alternativa, muestra una información detallada acerca de la prioridad de selección del paquete nombrado.
- **--important:** Muestra sólo las dependencias importantes.
- **dotty** pkg…**:** Toma una lista de paquetes de la línea de órdenes y genera una salida apropiada a través del paquete GraphViz[3]. El resultado será un conjunto de nodos y uniones representando las relaciones entre los paquetes. De forma predeterminada, los paquetes proporcionados mostrarán todas sus dependencias, lo que puede producir un grafo muy grande. Las líneas azules son pre-dependencias, las verdes son conflictos, las amarillas son sugeridas, naranja son recomendados, rojas son reemplazos, y las negras son dependencias. Atención, dotty no está capacitado para realizar gráficos de grandes listas de paquetes [7].

Esta es una aplicación bastante completa y cumple con la mayoría de los requisitos especificados, por lo que se tomará en cuenta para la propuesta de solución a emplear.

[3] GraphViz: Paquete que permite la creación y visualización de gráficos de líneas.

**Apt-rdepends**

Esta aplicación de consola puede enumerar por vía recursiva las dependencias de paquetes, ya sea hacia adelante o en reversa, un proceso muy similar a “Apt-cache”, además, puede modelar gráficos. “Apt-rdepends” realiza búsquedas a través de la caché del APT para encontrar las dependencias de dicho paquete, por omisión, “apt-rdepends” muestra una lista de cada dependencia que posee el paquete además de mostrar que cada uno de estos paquetes cumplen con sus dependencias, trabajando de manera recursiva. Está escrito en Perl y actualmente se encuentra por la versión 1.3.0.

Entre sus principales opciones se encuentran las siguientes:

- **-b,--build-depends:** Muestra las dependencias directas del paquete seleccionado.
- **-d,--dotty:** En modalidad dotty permite listar los paquetes de manera similar a “**apt-cache**”. El resultado será un conjunto de nodos y bordes que representan las relaciones entre los paquetes.
- **-p,--print-state:** Muestra la versión del paquete y su estado.

A pesar de todas sus utilidades, “apt-rdepends” no está exenta de problemas debido a que no emula “apt-cache” perfectamente. Entre los principales inconvenientes que tiene está el que no muestra información sobre paquetes virtuales, ni tiene conocimiento de los mismos cuando está en el modo de la dependencia inversa. También “apt-rdepends” no sabe cómo parar después de que haya alcanzado una cierta profundidad convirtiéndolo en algo tedioso [8].

## 1.3.4 Synaptic

Synaptic es un gestor de paquetes gráfico para APT, el sistema de gestión de paquetes de Debian. Synaptic simplifica la instalación, eliminación y actualización de paquetes DEB en Debian y derivados

como Ubuntu, automatizando acciones como la inclusión de dependencias de un paquete concreto. Synaptic instala paquetes sueltos, de listas (llamadas repositorios) de Internet y de CD o DVD. Puede manejar varios al mismo tiempo, y se encarga de buscar las dependencias, instalándolas automáticamente.

Además, busca actualizaciones de paquetes, y puede bajarlos e instalarlos de forma automática o bajo demanda.

Hay que añadir que Synaptic lista los paquetes por categorías, origen, si está instalado o si tiene actualización disponible, y ofrece información variada sobre los paquetes incluidos.

Synaptic proporciona las siguientes características:

- Instalar, eliminar, configurar, actualizar y descargar paquetes.
- Actualizar el sistema completo.
- Gestionar el repositorio de paquetes.
- Buscar los paquetes por nombre, descripción y otras propiedades de los paquetes.
- Filtrar la lista de paquetes conocidos, por estatus, sección o propiedades.
- Ordenar listados de paquetes por letra inicial, estatus, entre otras opciones.
- Visualizar toda la documentación en línea referida al paquete.

Su principal problema es que hay paquetes que no aparecen en los repositorios por defecto de Synaptic, y no siempre están actualizados [9].

## 1.4 Valoración del estudio realizado

Una vez concluido el estudio de las herramientas para el análisis de repositorios, se puede decir que: **Summon** es una potente herramienta para el análisis de paquetes y parte de sus características serán tomadas en cuenta, pero el uso de Entropy limita la aplicación en cuanto a rendimiento. **Reprepro** es considerada una herramienta para la creación de repositorios personalizados y no para el análisis de estos, aunque implemente mecanismos dentro de la configuración del mismo que están relacionados con lo que se desea. En el caso de las librerías **apt-cache** y **apt-rdepends** de **APT** ambas permiten mostrar las dependencias que posee cada paquete, una solución cuyos métodos serán tomados en cuenta para la solución a proponer, pero al igual que las anteriores no satisfacen todas las necesidades. Por último, **Synaptic** es un gestor de paquetes y la solución a implementar, aunque posea elementos relacionados con un gestor, no llega a ser uno.

Por lo antes expuesto se propone desarrollar una nueva aplicación, en vez de seleccionar algunas de las estudiadas, teniendo en cuenta las siguientes características de las aplicaciones anteriores:

- Conectarse al repositorio GNU/Linux Nova.
- Buscar la información de cada paquete.
- Detectar las inconsistencias que poseen los paquetes.

## 1.5 Metodología de desarrollo

Una metodología de desarrollo de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Es un acumulado de procedimientos, técnicas, artefactos y herramientas que guían a los desarrolladores en el proceso de realización de software [10].

Hasta hace poco el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos). Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del “buen hacer” de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Entre las metodologías de desarrollo se encuentran:

**Tradicionales:**

- RUP (Rational Unified Process)
- MSF (Microsoft Solution Framework)
- Win-Win Spiral Model
- Iconix

**Ágiles:**

- XP (eXtreme Programming)

- Scrum
- Crystal
- DSDM (Dynamic Systems Development Method)
- FDD (Feature Deriven Development)
- Extreme Modeling
- AUP (Agil Unified Process)

Debido a que el equipo de desarrollo está compuesto por pocos integrantes, el tiempo de desarrollo es relativamente corto y los requisitos son cambiantes se optó por el uso de una metodología ágil. Se escogió como metodología a utilizar una variación de la metodología “Proceso Unificado Ágil” (AUP por sus siglas en inglés) en unión con el modelo CMMI-DEV v 1.3 desarrollada en la UCI denominada AUP-UCI, la cual es idónea para la entrega de productos en cortos períodos de tiempo; además de satisfacer las necesidades del proceso.

AUP-UCI es una metodología flexible que no requiere de una gran cantidad de desarrolladores. Es concisa en el aspecto de la documentación, permitiendo generar solo la necesaria y no la especificada para cada flujo de trabajo como lo hace RUP. Está diseñada para trabajar en proyectos pequeños donde la atención se centra en las actividades que realmente son importantes. Permite el uso de herramientas de cualquier tipo, incluyendo aquí las de código abierto. Es fácil de manejar a través de herramientas de edición HTML sin necesidad de ser adaptada y es una metodología que se ajusta y aprovecha las ventajas que brindan las metodologías ágiles.

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

permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

- **Ejecución:** En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
- **Cierre:** En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

## 1.6 Lenguajes, tecnologías y herramientas de desarrollo utilizadas

Para el desarrollo de la solución propuesta se realizó un estudio en cuanto al lenguaje de programación, las tecnologías y las herramientas a utilizar, teniendo en cuenta las ventajas de cada una y que cumplan con las condiciones del sistema a desarrollar.

### 1.6.1 Lenguajes

Los lenguajes de programación son herramientas que nos permiten crear programas y software. Entre ellos se encuentran Delphi, Visual Basic, Pascal, Java. Una computadora funciona bajo control de un programa el cual debe estar almacenado en la unidad de memoria; tales como el disco duro. A continuación se describen los diferentes lenguajes a utilizar durante la solución a realizar:

**Python**

Es un lenguaje interpretado de programación, multiplataforma, flexible, su implementación está bajo la licencia de código abierto Python Software Foundation License. Python, al ser un lenguaje interpretado, implica ahorro de tiempo durante el desarrollo de un programa ya que no necesita de compilación. El intérprete puede usarse de modo interactivo, el cual hace fácil el experimentar con las características del lenguaje para probar funciones durante la etapa inicial de desarrollo [11].

Permite escribir programas muy compactos y legibles, permitiendo ser más pequeños que sus contrapartes en C u otros lenguajes por las siguientes razones:

- **Lenguaje Interpretado o de Script:** Es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje de máquina.
- **Funciones y bibliotecas:** Dispone de un gran cúmulo de funciones incorporadas en el propio lenguaje, permitiendo el tratamiento de cadenas de caracteres, números y archivos. Además, existen librerías que son importadas con facilidad en los programas para tratar temas específicos.
- **Detección dinámica del tipo de variable en tiempo de ejecución:** Se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo de valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.
- **Sintaxis clara:** Posee una sintaxis visual gracias a la notación (con márgenes) de obligado cumplimiento. Además, para distanciar las porciones de código se aconseja tabular, colocando un margen al código que iría dentro de una función o un bucle. Esto beneficia a que todos los

programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

- **Multiplataforma:** El intérprete de Python está disponible en multitud de plataformas (Solaris, GNU/Linux, DOS, Windows, OS/2, Mac OS) por lo que si no se utilizan bibliotecas específicas de cada plataforma el programa podrá correr en todos estos sistemas sin grandes cambios.
- **Orientado a Objeto:** La orientación a objetos es un paradigma de programación en el que los conceptos relevantes se trasladan a clases y objetos del programa. La ejecución del programa consiste en una serie de interacciones entre los objetos.

**Django**

Es un framework de desarrollo web de código abierto, escrito en Python, que implementa el patrón de diseño conocido como Modelo–vista–plantilla. Fue desarrollado para gestionar varias páginas y fue liberada al público bajo una licencia BSD en julio de 2005.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conexión y extensión de sus componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés *Don't Repeat Yourself*). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

Mantiene de forma rigurosa un diseño limpio en su propio código y facilita que el programador siga las mejores prácticas de desarrollo web en las aplicaciones que crea. Fomenta el bajo acoplamiento, la filosofía de programación que dice que las distintas partes de la aplicación deben ser intercambiables y deben comunicarse unas con otras mediante APIs claras y concisas.

Django es usado por muchas compañías para el desarrollo de sus aplicaciones empresariales por la rica cantidad de características con que cuenta. Fue iniciado por Jacob Kaplan-Moss y Jeremy Dunck cuando necesitaban tener un marco de trabajo común para la construcción de aplicaciones web de uso intensivo y lo hizo un proyecto de código abierto en 2005 [12]. Algunas de las características con que cuenta este framework son:

- **Sistema de mapeo de objetos relacionales (ORM):** Permite definir los modelos de datos enteramente en Python, además de que provee una rica y dinámica API para acceso a bases de datos, aunque también da la posibilidad de construir su propio código SQL.
- **Una interfaz automática de administración:** Provee un simple mecanismo para la creación de interfaces para administradores que deban agregar o actualizar contenido de su sitio.
- **Sistema de plantillas:** Provee un mecanismo de plantillas poderoso, extensible y sin limitaciones específicas del framework para la separación del diseño, el contenido y el código de Python.
- **Sistema de cacheo:** Provee APIs para el cacheo de objetos en memoria usando Memcached u otras aplicaciones de cacheo para obtener un mejor rendimiento de la aplicación.
- **Internacionalización:** Tiene un completo soporte para aplicaciones en varios idiomas, dándole la posibilidad al desarrollador de tener el control total de sus cadenas en diferentes idiomas.
- **Gran cantidad de aplicaciones reusables y plugins:** La comunidad detrás del desarrollo del framework es inmensa, por lo que siguiendo los principios en los que está basado Django, se han desarrollado gran cantidad de aplicaciones y plugins listos para la producción de las tareas más comunes.

- **Soporte para múltiples bases de datos:** En la versión 1.2 fue introducida esta característica muy esperada por la comunidad, dando la posibilidad de tener varias bases de datos en un mismo proyecto de Django.

**UML**

Para el modelado de los elementos arquitectónicos fue utilizado el lenguaje UML que constituye el estándar para representación del desarrollo de software, ilustrando los procesos, objetos y entidades de un software, así como sus relaciones.

Las funciones de UML se pueden sintetizar en:

- **Visualizar:** Permite expresar de una forma gráfica un sistema, de manera que otra persona lo puedan entender.
- **Especificar:** Permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión [13].

### 1.6.2 Herramientas

Para el desarrollo de la aplicación se propone utilizar el entorno de desarrollo multiplataforma PyCharm en su versión 4.5.4, la cual incluye una gran cantidad de funcionalidades y características a utilizar en el

desarrollo de la solución destacando su integración con frameworks web como Django, cuya versión con la cual se va a trabajar es la 1.6.1.

Visual Paradigm en su versión 8.0 fue la herramienta CASE seleccionada, es una herramienta UML profesional que soporta el ciclo de vida completo de desarrollo del software: análisis y diseño, construcción, pruebas y despliegue. El lenguaje de modelado UML permite generación automática de código además de realizar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Para lograr consistencia y persistencia en los datos es necesario el empleo de bases de datos que respondan las necesidades de disponibilidad, estabilidad y velocidad de acceso. Para este caso se selecciona PostgreSQL en su versión 9.2.4, el cual es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD, la cual provee soporte para números de precisión arbitraria, cadenas de texto con un largo ilimitado, figuras geométricas (con una variedad de funciones asociadas), direcciones IP (IPv4 e Ipv6), bloques de direcciones estilo CIDR, direcciones MAC, "*arrays*". Este gestor cuenta con APIs para variedad de lenguajes entre los que se encuentran Java, Python, C++, C# entre otros.

## 1.7 Conclusiones parciales

En el presente capítulo fueron analizados diferentes sistemas encargados de la gestión de inconsistencias en repositorios, donde se identificó que no cumplen con las necesidades actuales para la distribución de GNU/Linux Nova. Por tanto, se decide desarrollar un nuevo sistema para la gestión de inconsistencias dentro de un repositorio. La metodología de desarrollo seleccionada fue AUP-UCI, la cual permite estructurar, planificar y controlar todo el proceso de desarrollo de la aplicación. Se seleccionó el lenguaje

de programación Python, usando el IDE[4] PyCharm 4.0.4 a través del framework Django 1.7.6 y como gestor de bases de datos se seleccionó PostgreSQL 9.2.4 debido a su compatibilidad con Django. También se hace uso de HTML y CSS.

[4] IDE: O Integrated Development Environment (en español *entorno de desarrollo integrado*) es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

# Capítulo 2: Análisis y diseño de la solución propuesta

## 2.1 Introducción

En el presente capítulo se exponen los requisitos del sistema, se realiza la modelación de los mismos en diagramas de casos de uso y su correspondiente descripción. Se elabora el diseño de la arquitectura del sistema y se explica el funcionamiento del sistema a través del diagrama de paquetes.

## 2.2 Propuesta de solución

Para dar solución a los objetivos planteados se desarrollará la herramienta DigRepo, la cual es un sistema disponible en plataforma web capaz de mostrar toda la información de los paquetes del repositorio GNU/Linux Nova además de ser capaz de detectar inconsistencias en los paquetes. Esto le brindará al usuario información de primera acerca del estado de los paquetes del repositorio, además de que en caso de encontrar alguna inconsistencia envía un reporte a los mantenedores del repositorio.

La clase principal se conectará al repositorio y descargará el fichero “Package” el cual contiene todos los paquetes, junto con sus respectivos metadatos. A continuación DigRepo se encarga de analizar la información que provee el fichero y posteriormente muestra la información. Con el objetivo de especificar la información del sistema de manera general se muestra a continuación el modelo de negocio.

## 2.3 Modelado del negocio

El modelado del negocio es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de

que el software desarrollado va a cumplir su propósito [14]. Por falta de un flujo de procesos bien definido y para una mejor comprensión de los procesos se decide realizar el modelo de dominio.

## 2.3.1 Modelo de dominio

El modelo de dominio es el encargado de representar los conceptos fundamentales para el desarrollo de una aplicación y las diferentes relaciones que existen entre ellos. Son presentados como clases los aspectos esenciales y las interrelaciones denotan una estructura de funcionamiento, brindan una mejor comprensión del medio actual para la concepción de un futuro sistema [15]. La siguiente ilustración muestra el modelo de dominio del sistema a implementar:

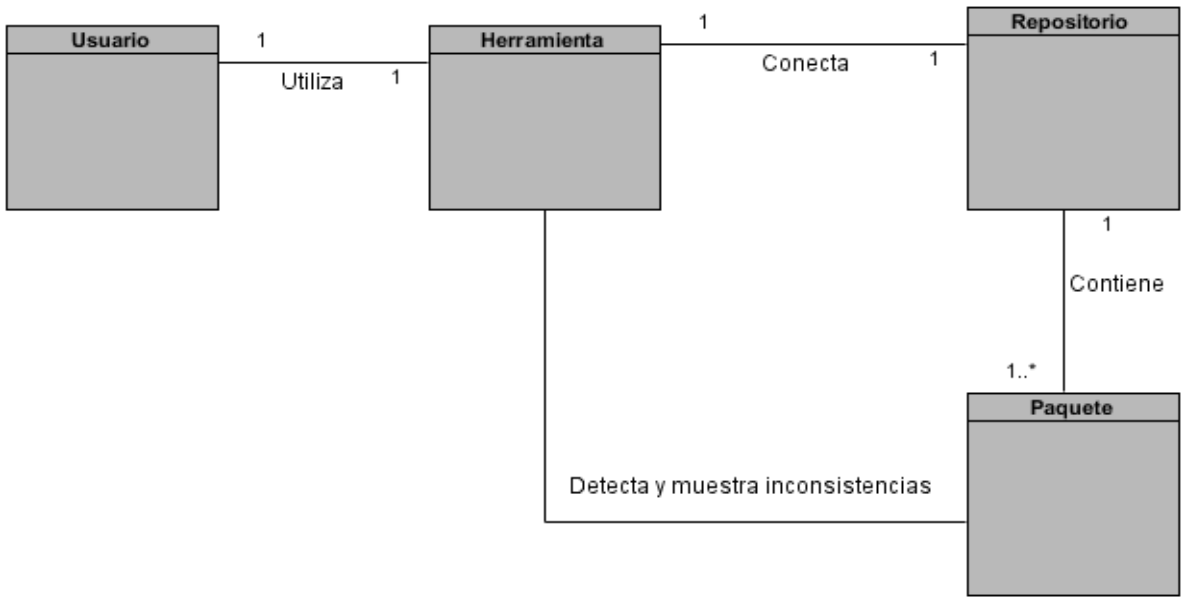


*Ilustración 1: Modelo conceptual.*

A continuación se describen cada uno de los objetos o relaciones que intervienen en el proceso de implementación:

*Tabla 2: Descripción de los objetos que intervienen en la implementación.*

| Objeto | Descripción |
|---|---|
| Usuario | Persona que interactúa con la aplicación encargada de buscar inconsistencias en el repositorio de GNU/Linux Nova. |
| Repositorio | Repositorio de la distribución GNU/Linux Nova. Está conformado por uno o por muchos paquetes. |
| Herramienta | Sistema a implementar. |
| Paquete | Paquete del repositorio. Contiene muchos metadatos. |

# 2.4 Requisitos

El esfuerzo principal en la disciplina requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. En el siguiente epígrafe se describirán los requisitos que se tuvieron en cuenta para elaborar la solución al problema planteado.

## 2.4.1 Requisitos funcionales

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer [16].

**Listado de requisitos funcionales**

**RF1.** Conectarse al repositorio.

**RF2**. Buscar paquete en el repositorio GNU/Linux Nova.

**RF3.** Listar los paquetes del repositorio con inconsistencias.

- **RF3.1** Listar todos los paquetes del repositorio con inconsistencias.
- **RF3.2.** Listar paquetes del repositorio con dependencias incumplidas.
- **RF3.3.** Listar paquetes del repositorio con inconsistencias de versión.
- **RF3.4.** Listar paquetes del repositorio que están incompletos.

**RF4.** Notificar a los mantenedores del repositorio.

## 2.4.2 Requisitos no funcionales

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad [17].

**Restricciones de diseño e implementación:**

- **RNF 1** Utilizar como lenguaje de programación Python.
- **RNF 2** Se utilizará PyCharm como IDE, Visual Paradigm como herramienta de modelado, UML en su versión 2.0 como lenguaje de modelado.
- **RNF 3** Se trabajará usando la arquitectura MTV (Model-Template-Plantilla).

**Interfaz de usuario**

- **RNF 4** Interfaz ajustada a la identidad marcaria XILEMA UCI.

## 2.5 Historias de usuario

Utilizadas para especificar los requisitos del software. En ellas se describen brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [18]. A continuación, se muestra la historia de usuario Conectarse al repositorio de paquetes de la distribución GNU/Linux Nova , el resto de las historias de usuario se muestran en el Anexo #2.

*Tabla 2: HU Conectarse al repositorio de paquetes de la distribución GNU/Linux Nova.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU1. | **Usuario:** Usuario. |
| **Nombre de Historia:** Conectarse al repositorio de paquetes de la distribución GNU/Linux Nova. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro. | |
| **Descripción:** El usuario introduce una dirección, si la misma es correcta el sistema se encargará de descargar los ficheros Package. En caso contrario se indica que la dirección es incorrecta. | |
| **Observaciones:** | |

## 2.6 Análisis y diseño

En esta disciplina, si se considera necesario, los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura). Además, en esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales. En el siguiente epígrafe se explicarán los principales elementos del análisis y el diseño tomados en cuenta para la implementación del sistema.

### 2.6.1 Arquitectura de software

De acuerdo al Software Engineering Institute (SEI), la arquitectura de software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos.

Los elementos pueden ser entidades que existen en tiempo de ejecución (objetos, hilos), entidades lógicas que existen en tiempo de desarrollo (clases, componentes) y entidades físicas (nodos, directorios). Por otro lado, las relaciones entre elementos dependen de propiedades visibles (o públicas) de los elementos, quedando ocultos los detalles de implementación. Finalmente, cada conjunto de elementos relacionados de un tipo particular corresponde a una estructura distinta, de ahí que la arquitectura está compuesta por distintas estructuras [19].

La arquitectura de software es de especial importancia ya que la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema. Ejemplos de atributos de calidad son el desempeño, que tiene que ver con el tiempo de respuesta del sistema a las peticiones que se le hacen, la usabilidad, que tiene que ver con qué tan sencillo les resulta a los usuarios realizar operaciones con el sistema. Los atributos de calidad son parte

de los requerimientos (no funcionales) del sistema y son características que deben expresarse de forma cuantitativa. No tiene sentido, por ejemplo, decir que el sistema debe devolver una petición “de manera rápida”, o presentar una página “ligera”, ya que no es posible evaluar objetivamente si el sistema cubre o no esos requerimientos.

La manera en que se estructura un sistema permitirá o impedirá que se satisfagan los atributos de calidad. Por ejemplo, un sistema estructurado de tal manera que una petición deba transitar por muchos componentes antes de que se devuelva una respuesta podría tener un desempeño pobre. Por otro lado, un sistema estructurado de tal manera que los componentes estén altamente acoplados entre ellos limitará severamente la modificación del mismo. Curiosamente, la estructuración tiene un impacto mucho menor respecto a los requerimientos funcionales del sistema. Por ejemplo, un sistema difícil de modificar puede satisfacer plenamente los requerimientos funcionales que se le imponen.

Además de los atributos de calidad, la arquitectura de software juega un papel fundamental para guiar el desarrollo. Una de las múltiples estructuras que la componen se enfoca en partir el sistema en componentes que serán desarrollados por individuos o grupos de individuos. La identificación de esta estructura de asignación de trabajo es esencial para apoyar las tareas de planeación del proyecto.

La herramienta DigRepo trabaja con la arquitectura comúnmente conocida como Modelo-Plantilla-Vista (por sus siglas en inglés MTV). Django fue diseñado para promover el acoplamiento débil y la estricta separación entre las piezas de una aplicación. Si se sigue esta filosofía, es fácil hacer cambios en un lugar particular de la aplicación sin afectar otras piezas. En las funciones de vista, por ejemplo, se discute la importancia de separar la lógica de negocios de la lógica de presentación usando un sistema de plantillas. Con la capa de la base de datos, se aplica esa misma filosofía para el acceso lógico a los datos.

- *M* significa "Model" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- *T* significa "Template" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web u otro tipo de documento.
- *V* significa "View" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre los modelos y las plantillas [20].

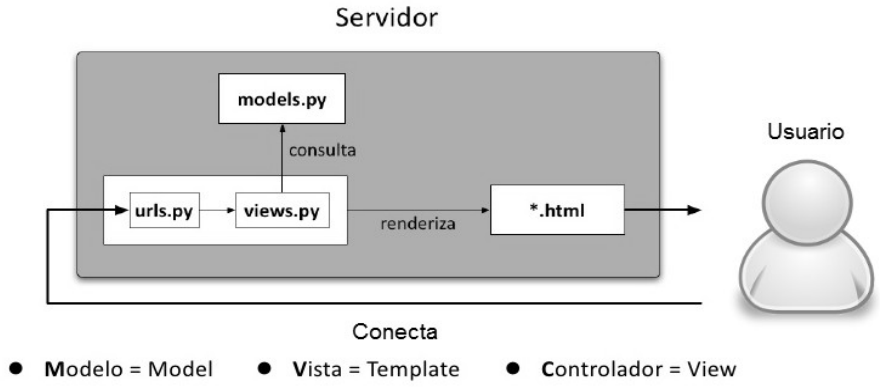


*Ilustración 2: Arquitectura MTV.*

## 2.6.2 Patrones de diseño

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. GRASP es un acrónimo que significa General Responsibility Assignment Software Patterns. El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos [21].

A continuación se explican los patrones utilizados

**Experto en información**

El GRASP de experto en información es el principio básico de asignación de responsabilidades. Indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). Ejemplo de ello es el método **Read ()**.

**Controlador**

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto se asigna de esa forma para aumentar la reutilización de código y a la vez tener un mayor control. Ejemplo de ello es **Home ()**.

**Bajo acoplamiento**

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de

clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. Un ejemplo de lo antes expuesto lo constituye la clase **Package**.

### 2.6.3 Diagrama de paquetes

Los diagramas de paquetes muestran las dependencias entre los paquetes que componen un sistema. Es decir, muestra cómo el mismo está dividido en agrupaciones lógicas y las dependencias entre esas agrupaciones [22].

- **DigRepo:** Contiene los elementos principales para llamar y mostrar toda la información provista.
- **Templates:** Formado por un conjunto de plantillas
- **Statics:** Almacena archivos estáticos.
- **connection:** Encargada de establecer la conexión con el sistema.
- **file_processor:** Es la encargada de la descarga y procesamiento del archivo *Package*. Se obtienen todos los metadatos necesarios.
- **package_module:** Contiene toda la información obtenida de los paquetes.
- **error_detection:** Contiene los métodos para la detección de las inconsistencias de paquetes.

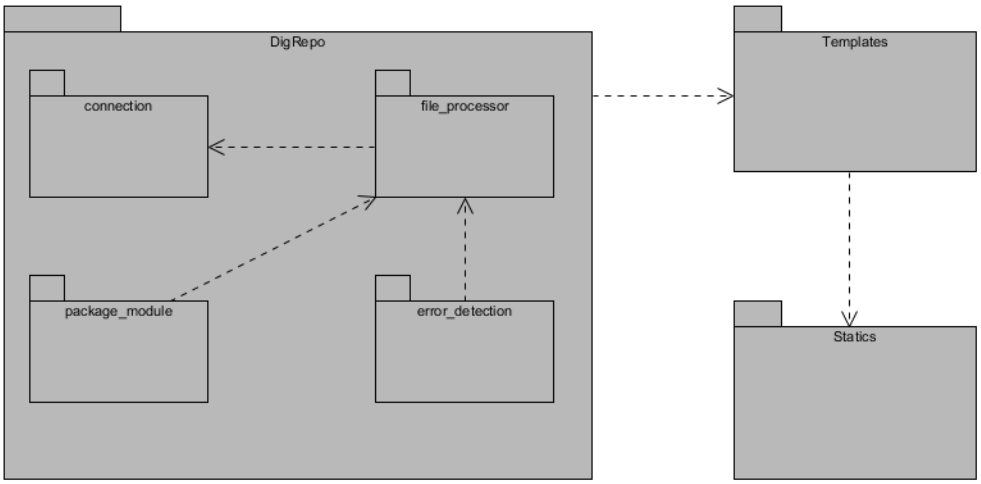


*Ilustración 3: Diagrama de paquetes.*

## 2.7 Conclusiones parciales

Como parte del proceso de análisis y diseño de la solución propuesta se realizó la extracción de los requisitos funcionales y no funcionales de la herramienta DigRepo a desarrollar. Se modeló, como parte de la metodología, el caso de uso correspondiente a la solución y el diagrama de paquete los cuales permiten representar parte de la lógica del negocio, y el modelo de dominio que permite representar las entidades relacionadas en la solución. Se hizo uso de la arquitectura MTV la cual permite representar la estructura de una manera similar con respecto a MVC así como de patrones de diseño como buenas prácticas durante la implementación.

# Capítulo 3: Implementación y Prueba

## 3.1 Introducción

Para un despliegue exitoso de DigRepo primero este debe pasar por un conjunto de pruebas las cuales permiten validar el correcto funcionamiento del sistema. Para ello en el siguiente capítulo se muestra la descripción de los principales métodos implementados; también se describe el diseño de las pruebas realizadas a la herramienta DigRepo y los resultados obtenidos durante cada iteración. Además se explican de manera breve los resultados arrojados por las pruebas realizadas a las diferentes funcionalidades.

## 3.2 Descripción de los principales métodos implementados

Para lograr un correcto funcionamiento de la aplicación se implementaron los siguientes métodos:

- **Conect ()**: Se conecta al repositorio de GNU/Linux Nova y descarga el fichero Package. En caso de no estar conectado a una red el sistema muestra un mensaje de error.
- **Read ()**: Se encarga de trabajar con el fichero Package, almacena los principales aspectos de la descripción de un paquete. Para ello lee línea a línea y almacena en la clase Package el nombre de cada paquete, su versión, así como una lista de dependencias.
- **Buscar ()**: En caso de existir el nombre dentro de la lista de paquetes previamente construida muestra todos los datos del paquete.
- **Search_Dep ()**: Realiza la búsqueda de inconsistencia de los paquetes previamente almacenados.

- **Listar_Errores ()**: Se encarga de mostrar al usuario las inconsistencias previamente encontradas.

## 3.3 Pruebas de Software

El desarrollo de cada sistema informático debe pasar por un conjunto de pruebas con el objetivo de comprobar si cumple con los requisitos definidos inicialmente y además validar su buen funcionamiento. Para el caso de la herramienta DigRepo se requiere realizar pruebas que demuestren que cada función es plenamente operacional, estas se denominan pruebas de caja negra. Otra variante es cuando se conoce el funcionamiento interno del producto, se aplican pruebas para asegurarse de que todas las piezas encajan, estas son las pruebas de caja blanca [23]. Durante todo el proceso de desarrollo se le realizaron al sistema pruebas unitarias, las que confirmaron la efectividad de las pruebas de caja blanca. A continuación se muestran los resultados arrojados por las pruebas de camino básico como una propuesta de caja blanca y posteriormente se muestran los resultados de las pruebas de aceptación.

### 3.3.1 Prueba del camino básico

El método del camino básico (propuesto por McCabe [24]) permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los componentes son:

**Nodo**

Cada círculo representado se denomina nodo del Grafo de Flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser también que hallan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

**Aristas**

Las flechas del grafo se denominan aristas y representan el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

**Regiones**

Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran. La cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo. Cuando en un diseño se encuentran condiciones compuestas (uno o más operadores AND, NAND, NOR lógicos en una sentencia condicional), la generación del grafo de flujo se hace un poco más complicada.

A continuación se muestra un ejemplo de prueba del camino básico al método ObtenerListaDepVer:

*Tabla 3: Código del método ObtenerListaDepVer.*

| Valor | Código |
|---|---|
| 1 | def ObtenerListaDepVer(request):<br>  lista_inicial = ''<br>  f = open('D:/Prueba.txt','r') |
| 2 | for line in f: |
| 3 | if line.__contains__('Depends: '):<br>  line = line.replace('Depends: ','').replace('(','').replace(')','').replace('= ','=').replace(',','') |
| 4 | lista_inicial = line |
| 5 | lista_intermedia = lista_inicial.split() #Automáticamente split divide por el espacio: ' '<br>  d = 0<br>  pos = 0<br>  array_final = [['' for i in range(2)] for i in range(lista_intermedia.__len__())] |
| 6 | for i in lista_intermedia: |
| 7 | if i.startswith('>') or i.startswith('='):<br>  array_final[d][1] = i<br>  d += 1<br>  pos += 1 |
| 8 | else:<br>  array_final[d][0] = i<br>  pos += 1 |
| 9 | if not lista_intermedia[pos].startswith('>'): |
| 10 |  lista_intermedia[pos].startswith('=') |

| | |
|---|---|
| 11 | d += 1 |
| 12 | return HttpResponse(array_final) |

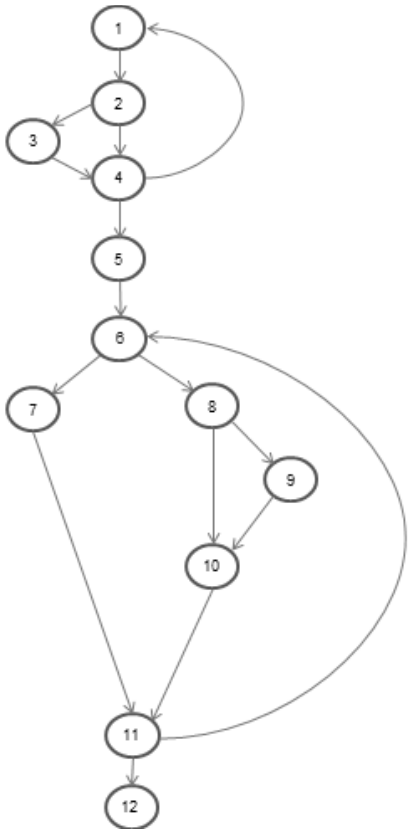


*Ilustración 4: Diagrama de flujo del método ObtenerListaDepVer.*

A partir del cálculo de la complejidad ciclomática se puede obtener el número de pruebas a realizar así como el número de posibles caminos:

V (G) = Aristas – Nodo + 2 = 16 – 12 + 2 = 6

V (G) = Regiones + 1 = 6

La representación de los caminos básicos son los siguientes:

Camino básico # 1: 1-2-4-5-6-7-11-12

Camino básico # 2: 1-2-3-4-5-6-7-11-12

Camino básico # 3: 1-2-4-5-6-7-8-10-11-12

Camino básico # 4: 1-2-4-5-6-7-8-9-10-11-12

Camino básico # 5: 1-2-3-4-5-6-7-8-10-11-12

Camino básico # 6: 1-2-3-4-5-6-7-8-9-10-11-12

Se diseñaron seis pruebas para este caso, y se ejecutaron satisfactoriamente, permitiendo concluir que la implementación de la funcionalidad correspondiente al escenario en el cual se realiza obtiene las dependencias con sus respectivas versiones de un paquete, para ser mostrado posteriormente en una interfaz, fue correctamente implementada.

Este mismo proceso para la determinación del número de pruebas, se realizó para cada requisito funcional, lo cual permitió constatar que todas las funcionalidades quedaron correctamente implementadas.

## 3.3.2 Pruebas de aceptación

Las pruebas de aceptación tienen como función validar que el sistema cumpla con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento, estas pruebas las realiza el cliente. Con respeto a las pruebas funcionales, se realizan sobre el sistema completo, y buscan una cobertura de la especificación de requisitos, no se realizan durante el desarrollo, pues sería impresentable para el cliente; son presentadas una vez pasada todas las pruebas de integración por parte del desarrollador.

Se emplean técnicas denominadas "pruebas alfa" y "pruebas beta". Las pruebas alfa consisten en invitar al cliente a que pruebe el sistema, se trabaja en un entorno controlado y el cliente siempre tiene un experto a mano para ayudarle a usar el sistema y para analizar los resultados. Las pruebas beta vienen después de las pruebas alfa, y se desarrollan en el entorno del cliente, un entorno que está fuera de control, aquí el cliente se queda solo con el producto y trata de encontrarle fallos (reales o imaginarios) de los que informa al desarrollador [25].

*Tabla 4: Caso de Prueba Buscar paquete.*

| **Caso de Prueba 1** | |
|---|---|
| **Nombre de Caso de Prueba:** Buscar paquete | |
| **Descripción de la Prueba:** Mostrar los datos del paquete. | |
| **Condiciones de Ejecución:** Conectarse al repositorio. | |
| **Entrada/Pasos de Ejecución:** | |
| 1. | Se presiona el botón “Buscar Paquete” desplegando un cuadro de texto. |
| 2. | Introduce el nombre del paquete. |
| 3. | Si coincide el nombre se muestra la información sobre el mismo. |
| **Resultado Esperado:** Se muestra en la interfaz el nombre del paquete junto con los datos disponibles del mismo. | |
| **Evaluación de la Prueba:** Satisfactoria | |

## 3.4 Resultados de las Pruebas de Aceptación

A continuación se identifican los resultados arrojados según las pruebas de aceptación realizadas a la aplicación, hasta el momento se han realizado solamente 3 iteraciones de pruebas:

- En la primera iteración se obtuvo como resultado un total de 12 No Conformidades (NC), de ellas 8 Significativas (S), 4 No Significativas (NS) y 2 Recomendaciones (R).
- La segunda iteración de pruebas arrojó como resultado 7 NC, de ellas 5 S, 2 NS y 4 R.
- En la tercera y última iteración de pruebas se obtuvo como resultado 3 R solamente.

La siguiente gráfica muestra los resultados obtenidos para cada una de las iteraciones:

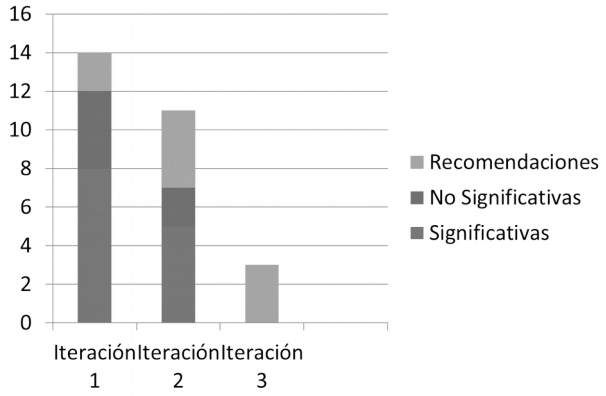


*Ilustración 7: Resultados de las pruebas de aceptación.*

## 3.5 Conclusiones parciales

A lo largo de este capítulo se realizó la descripción de los casos de prueba basados en requisitos utilizados durante las iteraciones de pruebas a la herramienta DigRepo. Para realizar la verificación del sistema se decidió utilizar las técnicas de caja blanca y pruebas de aceptación, para darle un mejor acabado al producto que se deseaba obtener siguiendo los requerimientos del sistema. Se detectaron y corrigieron una serie de no conformidades gracias a la realización del proceso de pruebas, validando así el correcto funcionamiento del sistema.

# Conclusiones

Con la culminación del presente trabajo de diploma se cumplieron cada uno de los objetivos trazados, distinguiéndose de manera general los siguientes aspectos:

1. El estudio de sistemas homólogos para la gestión de repositorios permitió definir una solución capaz de detectar inconsistencias al repositorio GNU/Linux Nova y enviar un reporte a los mantenedores del repositorio con las mismas.
2. La solución implementada se construyó siguiendo las bases de la arquitectura MTV, la cual permitió definir características y bases importantes a la misma. El uso de patrones contribuyó a un acabado óptimo de la solución.
3. Se detectaron en total una decena de inconsistencias de paquetes las cuales serán notificadas a los mantenedores de repositorio para la distribución GNU/Linux Nova.
4. La herramienta facilitó el proceso de detección de inconsistencias, contribuyendo así con el mantenimiento del repositorio para la distribución GNU/Linux Nova.
5. Las pruebas realizadas al producto implementado garantizaron la usabilidad del mismo y evidenciaron que cumple con los requerimientos definidos al inicio de la investigación, estas permitieron la corrección de las no conformidades encontradas.

# Recomendaciones

1. Añadirle a la herramienta una funcionalidad para que, además de detectar inconsistencias, pueda reparar las mismas.
2. Incorporar un mecanismo en la herramienta que permita determinar el índice de calidad del repositorio y su grado de evolución con respecto a la distribución padre.

# Bibliografía

Larman, Craig. UML y Patrones. Segunda edición, 2003.

S. Pressman, Roger. Ingeniería de software. Un enfoque práctico. Quinta edición, 2001.

T. Rodríguez Sánchez, Metodología de desarrollo para la Actividad productiva de la UCI.

Arthur H. Watson; Thomas J. McCabe*,* Las pruebas estructuradas: Una prueba metodológica usando la métrica de complejidad ciclomática.

Canós, José H., Letelier, Patricio y Penadés, M. Carmen. Metodologías Ágiles en el Desarrollo de Software. Universidad Politécnica de Valencia. Valencia: s.n., 2011.

# Referencias bibliográficas

[1] La Definición de Software Libre -Proyecto GNU -Free Software Foundation (FSF), [no date]. [online], [Accessed 8 Octubre 2015]. Available from: http://www.gnu.org/philosophy/free-sw.es.html

[2] ¿Qué es exactamente un Paquete de software? | MicroTeknologias, [no date]. [online], [Accessed 11 Octubre 2015]. Available from: https://microteknologias.wordpress.com/2009/03/13/%C2%BFque-es-exactamente-un-paquete-de-software/

[3] PuppyLinux: Dependencias, [no date]. [online], [Accessed 11 Octubre 2016]. Available from: http://puppylinux.org/wikka/es_Dependencias

[4] Significado de Inconsistencia - Qué es, Definición y Concepto, [no date]. [online], [Accessed 11 Octubre 2015]. Available from: http://quesignificado.com/inconsistencia/

[5] TRABAJO DE DIPLOMA - TD_1278_08.pdf, [no date]. [online], [Accessed 6 April 2016]. Available from: http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_1278_08/1/TD_1278_08.pdf

[6] reprepro, [no date]. [online], [Accessed 16 Octubre 2015]. Available from: https://mirrorer.alioth.debian.org/

[7] Ubuntu Manpage: apt-cache - query the APT cache, [no date]. [online], [Accessed 25 Octubre 2015]. Available from: http://manpages.ubuntu.com/manpages/precise/en/man8/apt-cache.8.html

[8] Ubuntu Manpage: apt-rdepends - performs recursive dependency listings similar to, [no date]. [online], [Accessed 25 Octubre 2015]. Available from: http://manpages.ubuntu.com/manpages/hardy/man8/apt-rdepends.8.html

[9] Synaptic (Linux) - Descargar, [no date]. [online], [Accessed 26 Octubre 2015]. Available from: http://synaptic.softonic.com/linux

[10] T. Rodríguez Sánchez, Metodología de desarrollo para la Actividad productiva de la UCI.

[11] Características de Python, [no date]. [online], [Accessed 13 Noviembre 2015]. Available from: http://dev.laptop.org/~edsiper/byteofpython_spanish/ch01s02.html

[12] Django, [no date]. [online], [Accessed 15 Noviembre 2015]. Available from: https://docs.djangoproject.com/

[13] 6.qxd - 026067.pdf, [no date]. [online], [Accessed 15 Noviembre 2015]. Available from: http://www.acta.es/medios/articulos/informatica_y_computacion/026067.pdf

[14] S. Pressman, Roger. Ingeniería de software. Un enfoque práctico. Quinta edición, 2001.

[15] S. Pressman, Roger. Ingeniería de software. Un enfoque práctico. Quinta edición, 2001.

[16] Requerimientos funcionales y no funcionales, [no date]. [online [Accessed 12 February 2016]. Available from: https://es.scribd.com/doc/37187866/Requerimientos-funcionales-y-no-funcionales

[17] Requerimientos Funcionales y No Funcionales (RF/RNF), [no date]. [online], [Accessed 12 February 2016]. Available from: http://ingenieriadesoftware.bligoo.com.mx/requerimientos-funcionales-y-no-funcionales-rf-rnf

[18] Can'os, José H., Letelier, Patricio y Penadés, M. Carmen. Metodologías Ágiles en el Desarrollo de Software. Universidad Politécnica de Valencia. Valencia: s.n., 2011.

[19] Arquitectura de Software | SG, [no date]. [online], [Accessed 17 February 2016]. Available from: http://sg.com.mx/revista/27/arquitectura-software

[20] Arquitectura de Sistemas Informáticos: Arquitectura de Django framework (parte 1), [no date]. [online], [Accessed 21 February 2016]. Available from: http://metodologiasdesistemas.blogspot.com/2010/10/arquitectura-del-django-framework-parte.html

[21] Patrones GRASP | Prácticas de Software, [no date]. [online], [Accessed 21 February 2016]. Available from: http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/

[22] Larman, Craig. UML y Patrones. Segunda edición, 2003

[23] *ElevenPaths Blog: QA: Pruebas para asegurar la calidad del producto software (III), [no date]. [online], [Accessed 1 April 2016]. Available from: http://blog.elevenpaths.com/2014/12/qa-pruebas-para-asegurar-la-calidad-del.html*

[24] S. Pressman, Roger. Ingeniería de software. Un enfoque práctico. Quinta edición, 2001.

[25] La prueba de aceptación es la prueba más importante para los productos software, [no date]. [online], [Accessed 1 April 2016]. Available from: http://www.pruebasdesoftware.com/pruebadeaceptacion.htm

# Anexos

## Anexo #1: Encuesta realizada a estudiantes y trabajadores acerca de la funcionalidad del repositorio de la distribución GNU/Linux Nova

Esta encuesta fue diseñada por la dirección del centro de CESOL, con el objetivo de conocer las opiniones acerca del funcionamiento del repositorio destinado a la distribución GNU/Linux Nova. Les pedimos de favor que sean sinceros a la hora de responder ya que sus opiniones permitirán mejorar los servicios prestados a los usuarios.

**Pregunta 1:**

¿Está satisfecho con los paquetes actuales que dispone el repositorio? En caso contrario exponga cuáles deberían incluirse y por qué.

**Pregunta 2:**

¿Ha encontrado problemas a la hora de instalar algún paquete? En caso afirmativo diga cuál o cuáles son.

**Pregunta 3:**

¿Considera que se deben realizar cambios en el repositorio? En caso afirmativo explique por qué, y de ser posible indique brevemente cómo debería organizarse el mismo.

**Pregunta 4:**

¿Tiene alguna duda o sugerencia que hacer?

**Gracias por responder.**

## Anexo #2:Tablas de descripción de historias de usuarios

*Tabla 5: HU Conectarse al repositorio de paquetes de la distribución GNU/Linux Nov.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU1. | **Usuario:** Usuario. |
| **Nombre de Historia:** Conectarse al repositorio de paquetes de la distribución GNU/Linux Nova. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro. | |
| **Descripción:** El usuario introduce una dirección, si la misma es correcta el sistema se encargará de descargar los ficheros Package. En caso contrario se indica que la dirección es incorrecta. | |
| **Observaciones:** | |

*Tabla 6: HU Buscar paquete en el repositorio GNU/Linux Nova.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU2. | **Usuario:** Usuario. |
| **Nombre de Historia:** Buscar paquete en el repositorio GNU/Linux Nova. | |
| **Prioridad en Negocio:** Media. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro | |
| **Descripción:** El usuario introduce el nombre de un paquete, si el nombre del mismo es correcto el sistema muestra toda la información disponible del mismo: su nombre, las dependencias que posee, la versión en la que se | |

| |
|---|
| encuentra, la descripción del paquete y las inconsistencias que pueda tener. En caso contrario se indica que el nombre es incorrecto o no se encuentra en el repositorio. |
| **Observaciones:** |

*Tabla 7: HU Listar todos los paquetes del repositorio con inconsistencias.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU3. | **Usuario:** Usuario. |
| **Nombre de Historia:** Listar todos los paquetes del repositorio con inconsistencias. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Alta. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro | |
| **Descripción:** El usuario escoge la opción de buscar todos los paquetes del repositorio que posean algún tipo de inconsistencia. En caso de que no haya ninguno se indica que en el repositorio no hay inconsistencias. | |
| **Observaciones:** | |

*Tabla 8: HU Listar todos los paquetes con dependencias incumplidas.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU4. | **Usuario:** Usuario. |
| **Nombre de Historia:** Listar todos los paquetes con dependencias incumplidas. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro | |
| **Descripción:** El usuario escoge la opción de buscar todos los paquetes del repositorio que posean específicamente dependencias incumplidas. En caso de que no haya ninguno se indica que en el repositorio no hay inconsistencias | |

| |
|---|
| del tipo dependencias incumplidas. |
| **Observaciones:** |

*Tabla 9: HU Listar todos los paquetes incompletos.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU5. | **Usuario:** Usuario. |
| **Nombre de Historia:** Listar todos los paquetes incompletos. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro | |
| **Descripción:** El usuario escoge la opción de buscar todos los paquetes del repositorio que estén incompletos. En caso de que no haya ninguno se indica que en el repositorio no hay inconsistencias del tipo paquetes incompletos. | |
| **Observaciones:** | |

*Tabla 10: HU Listar todos los paquetes con inconsistencias de versión.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU6. | **Usuario:** Usuario. |
| **Nombre de Historia:** Listar todos los paquetes con inconsistencias de versión. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro | |
| **Descripción:** El usuario escoge la opción de buscar todos los paquetes del repositorio que posean inconsistencias de versión. En caso de que no haya ninguno se indica que en el repositorio no hay inconsistencias del tipo dependencias incumplidas. | |
| **Observaciones:** | |

*Tabla 11: HU Generar reporte de las inconsistencias encontradas.*

| **Historia de Usuario** | |
|---|---|
| **Numero:** HU7. | **Usuario:** Usuario. |
| **Nombre de Historia:** Generar reporte de las inconsistencias encontradas.. | |
| **Prioridad en Negocio:** Alta. | **Puntos Estimados:** |
| **Riesgo en Desarrollo:** Media. | **Iteración Asignada:** |
| **Programador Responsable:** Daniel Escarret de Lázaro | |
| **Descripción:** El usuario escoge la opción de enviar un reporte a los mantenedores del repositorio con las inconsistencias encontradas. Para ello puede enviarlo directamente o configurar los parámetros del servicio de correo SMTP: servidor, puerto y destinatarios y posteriormente puede enviar el reporte. En caso de que haya dejado algún campo vacío o incorrecto el sistema le alerta con el error correspondiente. | |
| **Observaciones:** | |