UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

CENTRO DE SOFTWARE LIBRE

DEPARTAMENTO DE SISTEMA OPERATIVO

PROCEDIMIENTO DE GESTIÓN DE LOS REPOSITORIOS DE LA DISTRIBUCIÓN CUBANA DE GNU/LINUX NOVA

Trabajo final presentado en opción al título de Máster en Informática Avanzada

Autor: Ing. Juan Manuel Fuentes Rodríguez

Tutores: Dr.C Yanio Hernández Heredia

Msc. Allan Pierra Fuentes

La Habana

2018

Agradecimientos

A mis tutores por su apoyo en la realización de la tesis.

A todo el equipo de desarrollo de Nova por estar siempre dispuestos a seguir mis ideas.

A todos los que de alguna forma u otra me dieron la mano para terminar esta investigación.

Dedicatoria

A mis bebés.

Declaración Jurada de Autoría

Declaro por este medio que yo Juan Manuel Fuentes Rodríguez, con carné de identidad 88081528126, soy el autor principal del trabajo final de maestría _ PROCEDIMIENTO DE GESTIÓN DE LOS REPOSITORIOS DE LA DISTRIBUCIÓN CUBANA DE GNU/LINUX NOVA _, desarrollada como parte de la Maestría en Informática Avanzada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría

en La Habana a los ____ días del mes de _____ del año _____.

Resumen

Los repositorios de software constituyen el punto central del uso y desarrollo de las distribuciones de GNU/Linux, representando la vía oficial para que los usuarios puedan instalar software confiable y recibir actualizaciones y correcciones. Actualmente en la distribución cubana de GNU/Linux Nova la falta de estandarización en los procesos vinculados a los repositorios atentan contra la seguridad e integridad del servicio que se presta a los Órganos y Organismos de la Administración Central del Estado. La presente investigación realiza un estudio de los estándares, herramientas y procesos de gestión de repositorios de diferentes distribuciones, propone un procedimiento de gestión de repositorios que contempla las verificaciones de integridad y seguridad a los paquetes y metadatos del repositorio, y lo implementa utilizando la herramienta de gestión de repositorios Aptly. La implantación del resultado de esta investigación aumenta la confiabilidad y la seguridad de los repositorios de la distribución cubana Nova.

Abstract

The software repositories represent the central point of the use and development of the GNU/Linux distributions and provide the official way for users to install reliable software and receive updates and corrections. Currently in the Cuban GNU/Linux distribution of Nova the lack of standardization in processes linked to repositories threaten security and integrity of the service provided to the Organs and Bodies of the Central Administration of the State. The present investigation carries out a study of the standards, tools and processes of management of repositories of different distributions, proposes a management procedure for repositories that contemplates integrity and security checks on packages and metadata of the repository, and implements it using the Aptly repository management tool. The implementation of the results of this research increases the reliability and safety of repositories of the Cuban distribution.

Tabla de contenido

Introducción. 9

Capítulo 1. Fundamentación teórica de la gestión de repositorios de paquetes en las distribuciones de GNU/Linux. 13

1.1 Características de los repositorios de paquetes en GNU/Linux. 13

1.1.1 Paquetes de software 13

1.1.1.1 Paquetes binarios 13

1.1.1.2 Paquete de código fuente 14

1.1.1.3 Dependencias entre paquetes 16

1.1.2 Repositorios de paquetes en Debian. 16

1.1.3 Estructura de los repositorios de paquetes de Nova. 17

1.1.4 Especificaciones de seguridad de los repositorios de paquetes. 18

1.2 Verificación de dependencias de los repositorios de paquetes de Debian. 20

1.2.1 Proyecto EDOS. 20

1.2.2 Dose3. 21

1.3 Gestión de repositorios de paquetes basados en Debian. 21

1.3.1 Proceso de gestión de los repositorios de paquetes en Debian. 22

1.3.2 Proceso de gestión de los repositorios de Ubuntu. 24

1.3.3 Proceso de gestión de repositorios de Nova. 26

1.3.3.1 Nova distributed build system (ndbs). 26

1.3.3.2 Proceso actual de gestión de repositorios de paquetes en Nova. 27

1.4 Herramientas de construcción y gestión de repositorios de software en Debian. 28

1.4.1 Debian Archive Kit (Dak). 28

1.4.2 Reprepro. 29

1.4.3 Aptly 31

1.5 Evaluación de las herramientas de gestión de repositorios de paquetes de distribuciones basadas en Debian. 34

1.6 Conclusiones parciales 33

Capítulo 2. Procedimiento de gestión de repositorios de paquetes de Nova. 34

2.1 Características de los repositorios de Nova. 39
2.2 Descripción general del proceso. 40
2.2.1 Indicadores del estado del repositorio recolectados en el procedimiento. 41
2.3 Componentes del procedimiento propuesto. 42
2.3.1 Subproceso Crear instantánea de software heredado. 43
2.3.2 Crear Instantánea de software propio 45
2.3.3 Subproceso Crear Instantánea de actualizaciones. 45
2.3.4 Filtrar software con soporte oficial. 46
2.3.5 Publicar ramas del repositorio. 47
2.3.6 Realizar verificación de integridad del repositorio. 48
2.3.7 Realizar verificación de dependencias 49
2.4 Diseño de la aplicación de software para el procedimiento. 49
2.4.1 Propuesta de solución. 49
2.4.2 Arquitectura del sistema. 50
2.4.3 Requisitos funcionales 51
2.5 Conclusiones parciales 53
Capítulo 3. Implementación y validación del modelo propuesto. 54
3.1 Implementación. 54
3.1.1 Lenguaje de programación. 54
3.1.2 Estándar de codificación. 54
3.1.3 Integración con el servidor de automatización e integración continua Jenkins. 55
3.2 Despliegue de la solución propuesta. 56
3.2.1 Diagrama de despliegue 56
3.2.2 Requerimientos de hardware. 56
3.2.3 Requerimientos de software. 56
3.3 Validación de la solución. 57
3.3.1 Validación funcional. Estudio de caso Nova Loongson. 57
3.3.1.1 Planteamiento de la hipótesis del estudio de caso. 57
3.3.1.2 Selección del proyecto piloto. 58

3.3.1.3 Identificación del método de comparación. 59
3.3.1.4 Planeación el estudio de caso. 59
3.3.1.5 Análisis de los resultados. 60
3.3.2 Evaluación de satisfacción grupal mediante el método Iadov. 61
3.4 Conclusiones parciales. 64
Conclusiones 65
Recomendaciones 66
Bibliografía 67
Anexos 72
Anexo 1: Lista de expertos y actores en la gestión de repositorios de software libre. 72
Anexo 2: Lista de variables claves del análisis estructural. 73
Anexo 3: Ponderación de los criterios de evaluación para la selección de la herramienta para la gestión de repositorios de paquetes de Debian. 74
Anexo 4: Cubrimiento funcional de los criterios de evaluación. 75
Anexo 5: Evaluación de las herramientas de gestión de repositorios aplicando QSOS. 76
Anexo 6: Encuesta para valorar la satisfacción de los clientes con el sistema de gestión de repositorios. 77

Introducción.

Como parte de la política de informatización de la sociedad trazada por el gobierno cubano se lleva a cabo un proceso de migración a estándares abiertos de las estaciones de trabajo en los Órganos y Organismos de la Administración Central del Estado (OACE), con el objetivo de disminuir la dependencia de Microsoft Windows y el software privativo en general. Todo este proceso está sostenido por la distribución cubana de GNU/Linux Nova, un sistema operativo de propósito general basado en Ubuntu desarrollado para satisfacer los requerimientos derivados del mencionado proceso de migración (Albalat, Fírvida, García, Machín 2012).

Para asegurar un correcto proceso de migración a software libre, Nova se desarrolla teniendo en cuenta los siguientes fundamentos (Pierra Fuentes 2011):

- Seguridad: mediante la utilización del modelo de desarrollo colaborativo que propone el movimiento de software libre y el acceso al código fuente, así como su exhaustivo proceso de revisión y auditoría, se asegura un sistema seguro de ataques y sin puertas traseras conocidas.
- Socio-adaptabilidad: brindar un sistema operativo hecho por cubanos y para los cubanos, alineado a las políticas que orienta la informatización nacional y optimizado para las condiciones tecnológicas del país, asegurando una mejor adopción de Nova en los OACE.
- Soberanía tecnológica: gracias a la asimilación de las tecnologías utilizadas, se obtiene la capacidad decisional sobre el uso y desarrollo de estas, y la posibilidad de evolucionar de forma autónoma.
- Sostenibilidad: mantener un proceso flexible y versátil, en constante innovación y consonancia con las nuevas tendencias tecnológicas, garantizando modelos de comercialización que permitan el ingreso de divisas por el concepto de exportación de productos y servicios.

La creación de la distribución de GNU/Linux Nova se realiza mediante la instalación de un conjunto de paquetes de software definidos en un sistema de archivo virgen (Pérez Herrera 2013). Estos paquetes, junto con todo el software compatible con la distribución, está almacenado en una estructura de archivos llamada repositorio, el cual constituye la vía oficial para que los usuarios accedan a nuevas aplicaciones para satisfacer sus necesidades, buscar actualizaciones y obtener correcciones de seguridad y de errores (Ubuntu 2017). El repositorio de Nova está firmado digitalmente para que los sistemas instalados puedan identificar cuando el software es procedente de las fuentes oficiales, aumentando así la confiabilidad del servicio prestado a los OACE.

Cada versión de Nova cuenta con repositorio destinado en el que son almacenados todos los paquetes compatibles con esta (Pierra Fuentes 2011). Este repositorio es creado a partir de la

liberación de la última versión LTS[1] de Ubuntu, con el objetivo de reutilizar todo el software posible con un periodo de soporte largo por comunidades internacionales. Para modificar o adaptar un paquete de la distribución, los desarrolladores descargan desde este repositorio el código fuente, lo compilan, construyen de los paquetes binarios y entregan los resultados al equipo de administración del repositorio para su inclusión en el mismo. Este proceso actualmente se realiza de forma personal (por la red o mediante el uso de medios extraíbles) lo que conlleva a sensibles riesgos de seguridad como la fuga de información sensible del desarrollo, pérdida o corrupción de los paquetes por el mal funcionamiento del medio de transporte y reducción de la eficiencia en el desempeño laboral de los desarrolladores.

Toda la responsabilidad de la gestión de los repositorios de Nova recae sobre los administradores del mismo, los cuáles cumplen sus obligaciones mediante la ejecución de *scripts* desarrollados por ellos mismos que pudieran no estar alineados con las políticas de desarrollo de Nova, lo que representa un riesgo debido a que son ejecutados con los permisos suficientes para poder corromper (intencionalmente o por error) todo el repositorio. Agrava esta situación el hecho de que los administradores del repositorio no realizan verificaciones de consistencia en el mismo, posibilitando la gestión de un repositorio incompleto con aplicaciones ausentes, paquetes con una suma de verificación incorrecta o con dependencias insatisfechas.

Los repositorios de Nova desde la versión 3.0 son gestionados mediante *reprepro*, una herramienta desarrollada para mantener de forma sencilla repositorios locales de paquetes Debian y réplicas de repositorios remotos (Link 2013), pero que contiene numerosos defectos que dificultan el trabajo de los especialistas y comprometen la integridad de los repositorios cuando es escalado a varios repositorios de gran tamaño como los de Nova.

A través del tiempo los administradores de *reprepro* en Nova han detectado que uno de los puntos más vulnerables es su base de datos. Actualmente esta herramienta hace uso de una base de datos local usando *Berkeley DB*, en la cual se almacena la información de todos los paquetes provistos por el repositorio, lo que constituye una decisión errónea según la ayuda oficial de la herramienta (Link 2015). Una vez iniciada una operación sobre el repositorio esta no debe ser interrumpida pues corrompería por completo la base de datos. A pesar de la existencia de procedimientos propios de la herramienta para este tipo de incidentes con la base de datos (Link 2011), en la mayoría de los casos estos no pueden ser resueltos y la única solución consiste en regenerar por completo los repositorios de Nova, perdiendo cuantioso tiempo y afectando la disponibilidad del servicio.

[1] Soporte de Largo Término por sus siglas en inglés

Otro problema de *reprepro* es que no permite la realización de operaciones simultáneas en el repositorio, que sumado a lo lento que resulta actualizar los índices de los repositorios grandes, disminuye considerablemente la eficiencia. Además de estos problemas, reprepro tiene ausencia de funcionalidades que pudieran incidir directamente en la calidad, seguridad y fiabilidad de los repositorios de Nova tales como:

- No brinda ningún mecanismo para retroceder a estados previos del repositorio, funcionalidad muy útil en casos de corrupción en la base de datos y para ocasiones en las que es necesario deshacer una mala operación con el repositorio.
- No posee una funcionalidad para realizar verificaciones de dependencias sobre el repositorio, por lo que resulta complejo saber qué paquete en el repositorio no puede ser instalado en una computadora con Nova.
- Es imposible tener en un mismo repositorio varias versiones de un mismo paquete.

Estos problemas se pueden resumir en que la falta de estandarización en los procesos de desarrollo de Nova que involucran la gestión de los repositorios, así como la utilización de una herramienta no idónea para el trabajo, acarrean problemas de seguridad, eficiencia y eficacia dentro del proyecto, lo que nos permite formular el siguiente **problema de la investigación**:

¿Cómo aumentar los niveles de seguridad y la confiabilidad en los procesos de gestión de los repositorios de software de Nova?

Para la resolución del problema científico se asume como **objeto de estudio** *los repositorios de software de las distribuciones de GNU/Linux*, teniendo como **campo de acción** *la gestión, especificaciones de seguridad y verificaciones de integridad de los repositorios de software de las distribuciones de GNU/Linux.*

Por lo que se propone como **objetivo general** de esta investigación:

Elaborar un procedimiento de gestión de los repositorios de software de Nova, que verifique la integridad de los paquetes publicados y que incorpore el uso de elementos criptográficos para aumentar la seguridad y confiabilidad del servicio.

Además, los siguientes **objetivos específicos**:

1. Realizar el estudio el estado del arte de la gestión de repositorios de software en las distribuciones de GNU/Linux en Cuba y el mundo.
2. Modelar los procesos para la gestión de repositorios de software de Nova.
3. Validar la propuesta a través de los métodos definidos en la investigación.

En la investigación se declara como **Hipótesis**:

La elaboración de un procedimiento para la gestión de los repositorios de software de Nova, que realice una verificación de integridad a los paquetes y que asegure la autenticidad mediante las firmas criptográficas, contribuirá a elevar la seguridad y confiabilidad de los repositorios de la distribución cubana de GNU/Linux.

La **novedad científica** de la investigación radica en la conceptualización y definición de los principales procesos envueltos en la gestión de los repositorios de Nova, estableciendo sus principios de seguridad, de acuerdo a las políticas trazadas por las entidades rectoras de la informatización en el país, e integridad. Se establece un procedimiento de gestión de repositorios automatizables en aras de aumentar la eficiencia del mantenimiento del repositorio, reducir el esfuerzo y estandarizar los procesos. Mediante la implementación y utilización de este procedimiento se logra la generación y el mantenimiento de los repositorios de Nova con mayores niveles de seguridad e integridad, empleando menor esfuerzo por parte del equipo de desarrollo.

Capítulo 1. Fundamentación teórica de la gestión de repositorios de paquetes en las distribuciones de GNU/Linux.

En este capítulo se realiza un estudio del estado del arte de los sistemas de gestión de repositorios utilizados por los desarrolladores de distintas distribuciones de GNU/Linux, con el objetivo de contar con un marco teórico que sirva de base para la obtención de la solución al problema planteado. Se exponen un conjunto de conceptos relacionados con los paquetes almacenados en estos, así como, así como las características que definen a los repositorios de paquetes y su gestión en las distribuciones de GNU/Linux

1.1 Características de los repositorios de paquetes en GNU/Linux.

Un repositorio de paquetes es un sitio donde estos son almacenados y mantenidos, pudiendo ser locales o remotos, dependiendo de si se encuentran en el sistema de archivos de la máquina del usuario que lo utiliza o si se encuentran en una localización remota accesible a través de la red, generalmente a través de los protocolos HTTP o FTP. Los repositorios de paquetes cuentan con archivos índices o metadatos que permiten la rápida localización de un paquete en el repositorio, así como la rápida obtención de la información del paquete sin tener que descargarlo o procesarlo (Fernández-Sanguino 2016). Para hacer uso de un repositorio, generalmente se utilizan programas llamados gestores de paquetes y facilitan la gestión de dependencias y las acciones comunes como instalar, actualizar, eliminar y buscar paquetes (Forbes, Stone, Parthasarathy, Toutonghi, Sliger 1998).

1.1.1 Paquetes de software

En términos informáticos un paquete de software consiste en un archivo comprimido utilizando un algoritmo de compresión conocido, que puede contener los elementos que integran una aplicación, así como los metadatos necesarios para su correcta identificación siguiendo un estándar definido (Fernández-Sanguino 2016). En las distribuciones de GNU/Linux los paquetes de software pueden ser de dos tipos: binarios o de código fuente.

1.1.1.1 Paquetes binarios

Un paquete binario es el conjunto de ejecutables, bibliotecas, archivos de configuración, documentación, licencias, etc. comprimidos que conforman una aplicación informática y que pude ser instalado en un ordenador por un tipo especial de herramienta llamada gestor de paquetes

(Forbes, Stone, Parthasarathy, Toutonghi, Sliger 1998). En las distribuciones basadas en Debian un paquete binario es una colección de archivos ejecutables (binarios compatibles con el cargador de arranque de Linux o *scripts*) y archivos complementarios necesarios para la ejecución de la aplicación, y que son utilizados por la herramienta *dpkg*[2] para la instalación.

Los paquetes de Debian son archivos comprimidos con extensión *deb* (o *udeb* para los paquetes utilizados por el instalador de Debian) utilizando el método de compresión *ar* y contienen:

- Archivo *debian-binary* que indica la versión del formato del paquete de Debian.
- Archivo *control.tar.gz* que contiene todos los metadatos relacionados con el paquete, como nombre, versión, dependencias, etc. Esta información es utilizada por los gestores de paquetes como *apt* y *packagekit*.
- Archivo *data.tar.gz* que contiene todos los ficheros y directorios pertenecientes al paquete organizados con la misma estructura que serán instalados en el sistema.

1.1.1.2 **Paquete de código fuente**

Un paquete de código fuente (o paquete fuente) consiste en un paquete que contiene todos los elementos necesarios para la construcción de uno o varios paquetes binarios. Dentro de los elementos contenidos dentro de un paquete fuente se encuentra el código fuente de la aplicación, las recetas necesarias para la compilación y construcción de los paquetes binarios, así como los recursos necesarios para la ejecución de las aplicaciones contenidas (como configuraciones, imágenes, base de datos, etc.) (Hertzog, Mas 2015).

Para las distribuciones de GNU/Linux basadas en Debian, los paquetes de código fuente consisten, en su forma más simple, en tres archivos (Pérez Herrera 2013):

- Un archivo con extensión *dsc* que consiste en un pequeño archivo de texto plano que contiene una cabecera con formato RFC 2822 que describe al paquete fuente e indica que otros archivos forman parte de este. Este debe ser firmado con la firma del mantenedor para garantizar autenticidad.
- Un archivo comprimido con extensión *orig.tar.gz* que contiene el código fuente provisto por el desarrollador original. Los mantenedores de paquetes de Debian no deben modificar este archivo con el objetivo de poder luego verificar el origen y la integridad del paquete.

[2] Dpkg es la herramienta oficial para la gestión de paquetes de Debian.

- Un archivo comprimido con extensión *debian.tar.gz* que contiene todas las modificaciones hechas por los mantenedores de paquetes para adaptar el código fuente original a la distribución. Además, están presenten las recetas para la construcción del paquete fuente, así como las instrucciones necesarias para el momento de instalación.

Existen tres formatos para los paquetes de software de Debian (Dpkg Developers 2016):

1. Formato 1.0: consiste en un archivo *orig.tar.gz* asociado a otro *diff.tar.gz* donde se guarda un parche con todas las modificaciones hechas por los mantenedores o un solo archivo *tar.gz* (en cuyo caso el paquete se denomina nativo pues fue desarrollado específicamente para Debian). Todos los paquetes generados con este formato deben usar el método de compresión *tar.gz.*

2. Formato 2.0: este formato no es recomendado para el uso pues es la primera versión de los nuevos formatos de paquetes de código fuente. El comportamiento es similar al formato *3.0 quilt* excepto que no utiliza una lista explícita de parches.

3. Formato 3.0: representa el formato más usado y el recomendado por las políticas de mantenimiento de Debian. Se caracteriza por la posibilidad de escoger el tipo de compresión deseada para el paquete (Gzip, Bzip2, Lzma y Xz) y por excluir por defecto todos los archivos temporales y de VCS[3]. Existen cinco tipos de paquetes con formato 3.0, de ellos tres se consideran para su estudio (git y bzr se consideran experimentales y no son soportados actualmente por el sistema de construcción de Debian):

   a) *quilt*: Está constituido por al menos un archivo *orig.tar.*<*extensión*>, que representa el código del desarrollador del paquete (aunque puede contener varios de estos mediante la forma *orig-componente.tar.*<*extensión*>), y un archivo *debian.tar.*<*extensión*> que contiene todas las modificaciones. Los parches son gestionados mediante la herramienta *quilt*.

   b) *native*: es una extensión del formato 1.0 nativo que puede usar las características del 3.0.

   c) *custom*: es un formato especial que no representa un paquete fuente real, pero puede ser usado para crear paquetes con archivos arbitrarios.

3 VCS: Sistema de control de versiones, por sus siglas en inglés.

1.1.1.3 **Dependencias entre paquetes**

Las dependencias de un paquete son aquellos paquetes binarios que son necesarios que estén instalados para que este funcione correctamente en un hardware y junto a otro software específico. Existen dos tipos de dependencias en Debian: las de construcción que son necesarias para construir los paquetes binarios a partir sus correspondientes paquetes fuentes y las de ejecución que son necesarias para que se pueda ejecutar el paquete (Pérez Herrera 2013).

Las dependencias de ejecución se clasifican en (Jackson 2017):

- Dependencias: lista de paquetes que tienen que estar instalados obligatoriamente para que la aplicación funcione.
- Recomendaciones: lista de paquetes que no son necesarios que estén instaladas para que la aplicación funcione, pero son altamente recomendadas para poder explotar todas las funcionalidades del paquete.
- Sugerencias: lista de paquete que no son necesarios para que la aplicación funcione, a pesar de ser sugeridos no es recomendado instalarlos a menos que se sepa exactamente que se está haciendo.
- Conflictos: lista de paquetes que no pueden estar instalados juntos, debido a que proveen un archivo con la misma ruta, o un servicio que solapa un puerto necesario o porque entorpece el correcto funcionamiento.
- Incompatibilidad: tiene un efecto similar a los conflictos, pero rompe con la instalación de los paquetes de la lista. Este caso sucede cuando existen actualizaciones que no poseen compatibilidad con versiones previas.
- Paquete provisto: mediante este se introducen los paquetes virtuales, los cuales son paquetes ficticios que pueden ser provistos por varios paquetes con el objetivo de describir una aplicación o servicio genérico (por ejemplo, *web-browser* es un paquete virtual provisto por los paquetes Firefox, Chrome, etc.)

1.1.2 **Repositorios de paquetes en Debian.**

Los repositorios de las distribuciones basadas en Debian consisten en un conjunto de paquetes (binarios o de código fuente) e índices con la información básica de los paquetes y su ruta relativa y pueden ser utilizados a través del gestor de paquetes *apt*. Por lo general están organizados de acuerdo a las versiones de la distribución y a la clasificación de los paquetes según su origen (componentes), y están estructurados de la siguiente forma:

- Directorio *dists*: contiene los índices y metadatos de las diferentes versiones de la distribución y sus componentes.

- Directorio *pool*: contiene todos los paquetes del repositorio. Cada paquete se encuentra en una carpeta con la siguiente topología: **<a>/<b>/<c>**, donde **a** es el componente a la que pertenece el paquete, **b** es la letra inicial del nombre del paquete (o las cuatro primeras en caso de que empiece con *lib*) y **c** el nombre del paquete.

Dentro del directorio *dists* los principales archivos son:

- *Release*: información general del repositorio (arquitecturas, componentes, versión, etc.). Además, se encuentra una suma de verificación de cada uno de los archivos presentes en el repositorio de la distribución.
- *Release.gpg*: llave digital (pública) usada para firmar los índices del repositorio.
- *InRelease*: contiene la misma información provista por el archivo *Release* pero está firmado con la llave digital disponible en *Release.gpg*.
- *Contents*-<*arquitectura*>: lista de archivos contenidos dentro de los paquetes del repositorio. Existe uno por cada arquitectura soportada en el repositorio y generalmente son comprimidos debido a su gran tamaño.
- *Packages*: archivo con toda la información de los paquetes de un componente y arquitectura específica.

1.1.3 **Estructura de los repositorios de paquetes de Nova.**

Los repositorios de software de Nova cuentan con la misma estructura que los repositorios de Debian y Ubuntu, pero a diferencia de estas distribuciones cuenta solamente con dos componentes (ver Fig. 1):

- principal: componente en el que se almacena todo el software con soporte oficial de la distribución Nova. El filtro para la selección de los paquetes de este componente es generado a partir de las dependencias y las recomendaciones de los metapaquetes de los principales productos: Escritorio, Ligero y Servidor.

- extendido: componente en el que se almacena todo el software compatible con la distribución, pero que no cuenta con soporte oficial por parte de Nova. En este componente se encuentran muchas aplicaciones de software provenientes de desarrollos comunitarios o con propósito específico.

A diferencia de las distribuciones de las que deriva, Nova no clasifica los paquetes debido a su origen, sino a su nivel de soporte. Debido a esta razón es común localizar en el mismo componente aplicaciones privativas y de licencias abiertas.

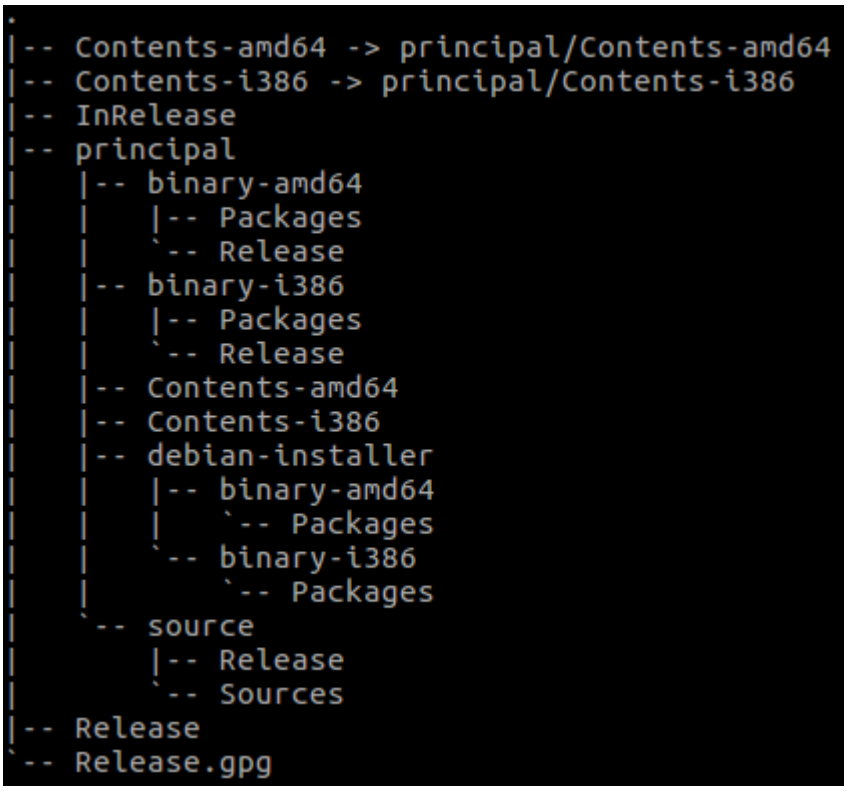


Fig. 1 Estructura del directorio dists en los repositorios de Nova.

1.1.4 **Especificaciones de seguridad de los repositorios de paquetes.**

Desde la versión 0.6 de *apt* se establece el mecanismo de validación criptográfica de los orígenes de los paquetes de Debian (y sus distribuciones derivadas), permitiendo que los usuarios puedan asegurarse de que los paquetes que instalan provienen de las fuentes oficiales de sus proveedores y no han sido manipulados mientras se transmitían por la red o almacenaban en las réplicas (Weimer 2005). Esta verificación está diseñada para contrarrestar las siguientes amenazas:

- Red de réplicas comprometida: una réplica del repositorio podría ser atacada con éxito y comenzar a entregar paquetes comprometidos a usuarios y otras réplicas.
- Ataques en la capa de red: el tráfico podría redirigirse y las respuestas DNS podrían falsificarse. Por lo tanto, un atacante podría atraer a un usuario (u otro espejo) para descargar paquetes comprometidos.

Para proveer un repositorio que cumpla con estas especificaciones se debe firmar el archivo *Release* con la firma del proveedor. Dentro de este archivo se enumeran todos los archivos *Packages* presentes en el repositorio, así como sus sumas de verificación (ver Fig. 2). Mediante esta especificación es posible establecer un mecanismo de verificación de los índices de los repositorios, solamente comprobando la confianza de la firma utilizada en el archivo *Release* y

verificando las sumas de verificaciones especificadas en este archivo contra las reales de los archivos *Packages* (Karachiwala 2017).

```
......
MD5Sum:
 d6e6d51bfe54969feec53795f56e061b 7315338 principal/binary-i386/Packages
 f07bc3e183fb61ccc45a16fe56d425df 1579947 principal/binary-i386/Packages.gz
 9d0a28c29339177e2dc6ff185343576d 1230559 principal/binary-i386/Packages.bz2
 90503596002fcbcdd122f2bbfcdc22b7 173 principal/binary-i386/Release
......
```

Fig. 2 Extracto del contenido del archivo Release.

Del mismo modo cada archivo *Packages* contiene la suma de verificación de los paquetes del repositorio. Estas políticas de seguridad para la verificación de integridad de los paquetes pueden ser probadas mediante una herramienta llamada *apt-secure*.

El esquema actual para la verificación de paquetes de un repositorio cuenta con los siguientes pasos (Fernández-Sanguino 2012):

1. El archivo *Release* incluye las sumas de verificación (MD5, SHA1, SHA256) de los archivos *Packages* (que a su vez contienen las sumas de verificación de los paquetes) es firmado utilizando una firma de una fuente confiable por parte de los administradores del repositorio.
2. El archivo *Release* firmado es descargado por *apt* y es almacenado junto con los archivos *Packages*.
3. Cuando se procede a la instalación de un paquete, este se descarga y su suma de verificación es generada y almacenada por *apt*.
4. El archivo *Release* es verificado (que posea una firma de confianza), se extrae las sumas de verificación del archivo *Packages* que es comprobada con la suma de verificación del mismo archivo almacenado localmente. Si es correcta se extrae la suma de verificación del paquete descargado.
5. Si la suma de verificación del paquete descargado coincide con la provista por el archivo *Packages* se instala. En caso contrario se deja almacenado en la cache para que el administrador del sistema decide qué hacer con él.

Por defecto, todas las herramientas para la gestión de repositorios con integración para firmas criptográficas generan repositorios que cumplen con estas especificaciones.

Es una práctica generalizada en las distribuciones basadas en Debian brindar las llaves públicas de los repositorios y paquetes de software contenidas en un paquete que viene instalado por defecto (este paquete tiene una prioridad muy alta por lo que es instalado desde el mismo sistema operativo base). En Nova este paquete es llamado *nova-keyring*. Si el usuario desea adicionar otras fuentes

no oficiales primero debe adicionar las llaves del repositorio en cuestión mediante la herramienta *apt-key* provista por *apt*.

1.2 Verificación de dependencias de los repositorios de paquetes de Debian.

La verificación de dependencias de los paquetes constituye un verdadero reto a la hora de mantener varios repositorios de una distribución de GNU/Linux pues, aunque en la actualidad los gestores de paquetes utilizados en Debian son capaces de resolver las dependencias de instalación y construcción en el lado del cliente, las herramientas de gestión de repositorios no explotan esta funcionalidad.

Un paquete *P* en un repositorio *D* de Debian se puede llamar instalable si y solo si existe un subconjunto *S* de *D* que contiene a *P* tal que para cada paquete de *S* se cumple que:

- Cada relación de dependencia (teniendo en cuenta todos los tipos de dependencias en los Debian, incluyendo paquetes virtuales) dentro de *S* es satisfecha por algún paquete de *S*.
- No existen conflictos con otros paquetes de *S* (Treinen 2014).

Para asegurar la calidad del repositorio es necesario verificar que cada uno de los paquetes sea instalable. En la actualidad existen varios programas para realizar la verificación de dependencias en un repositorio de paquetes de Debian.

1.2.1 Proyecto EDOS.

El proyecto EDOS fue creado con el propósito de aumentar la estabilidad de una distribución de GNU/Linux desde el punto de vista del administrador de los repositorios (Mancoosi 2006). Con este objetivo desarrollaron un conjunto de herramientas para rastrear problemas pertenecientes a los repositorios de paquetes de Debian.

Dentro de su conjunto de herramientas la más relevante para la investigación resulta *debcheck*, la cual toma como entrada un repositorio de paquetes de Debian y verifica si uno, varios o todos los paquetes del repositorio son instalables respecto a este, generando un reporte con el resultado de la operación (Mancinelli, Boender, Di Cosmo, Vouillon, Durak, Leroy, Treinen 2006). A pesar de los resultados del proyecto, el desarrollo de este proyecto está descontinuado desde el año 2013 en favor de la herramienta *dose3*, dejando de estar presente en los repositorios de Ubuntu desde la versión 14.04 y en Nova desde la versión 5.0.

1.2.2 **Dose3.**

La herramienta *dose3* es la sucesora del proyecto EDOS y brinda un conjunto de herramienta para verificar la instalabilidad de los paquetes en diferentes tipos de repositorios, incluyendo los de Debian. Dentro de sus herramientas las más relevantes para la investigación son *dose-builddebcheck* y *dose-distcheck*.

La herramienta *dose-distcheck* determina, para un conjunto de metadatos de los paquetes llamado repositorio, si los paquetes de este pueden instalarse en relación con el repositorio de acuerdo con las relaciones entre los paquetes declarados. Para Debian, estos metadatos deben estar en el formato establecido por *deb-control* separados por una línea en blanco (el mismo que el de los archivos *Packages* de un repositorio) y la instalabilidad de los paquetes es analizada de acuerdo a los campos *Depends*, *Pre-Depends*, *Conflicts*, *Break*, and *Provides* (Abate 2014a).

Por otra parte, la herramienta *dose-builddebcheck* tiene un comportamiento similar, pero se enfoca a determinar cuándo un entorno de compilación para los paquetes de código fuente puede ser instalado en una arquitectura nativa determinada utilizando un repositorio de paquetes binarios. Para esto, solo se tiene en cuenta los metadatos del paquete de código fuente: dependencias de construcción y conflictos de construcción, y las relaciones entre paquetes expresadas en el repositorio binario. La instalabilidad de los paquetes binarios se analiza de la misma forma que con la herramienta *dose-distcheck* (Abate 2014b).

Ambas herramientas generan sus reportes en formato YAML, pudiendo ser especificado cuales son elementos a reportar (las verificaciones fallidas o las correctas).

1.3 Gestión de repositorios de paquetes basados en Debian.

En la gestión de repositorios de paquetes para distribuciones basadas en Debian es necesario tener en cuenta los siguientes procesos:

- Creación de repositorios.
- Inclusión y actualización de paquetes de software heredado.
- Gestión de paquetes de software propio.
- Firma digital de los índices.
- Política de entrada del software.

1.3.1 **Proceso de gestión de los repositorios de paquetes en Debian.**

Con el objetivo de depurar el software y disminuir las vulnerabilidades de seguridad en los paquetes, Debian cuenta constantemente con tres versiones[4] principales en mantenimiento activo que definen el ciclo de vida de sus repositorios:

1) *unstable*: no se considera una versión oficial, sino como la versión en desarrollo de Debian y tiene como nombre clave *sid*. Esta versión contiene las últimas versiones de los paquetes del repositorio de Debian, por lo que es propensa a múltiples errores de toda índole y a dependencias fallidas debido al estado cambiante de sus paquetes (Yang 2017).

2) *testing*: esta versión constituye el estado actual de desarrollo de la próxima versión estable de Debian.

3) *stable*: contiene los paquetes de la versión estable de Debian. Este repositorio empieza como una copia de los paquetes estables de la versión anterior.

El principio general para la gestión de los repositorios de paquetes de software de Debian se basa en las políticas definidas para mover los paquetes de software entre las versiones de mantenimiento en dependencia de su grado de estabilidad.

[4] Debian cuenta además con otras distribuciones (experimental, oldstable, backports, etc) que son utilizadas en el proceso de gestión para flujos secundarios.

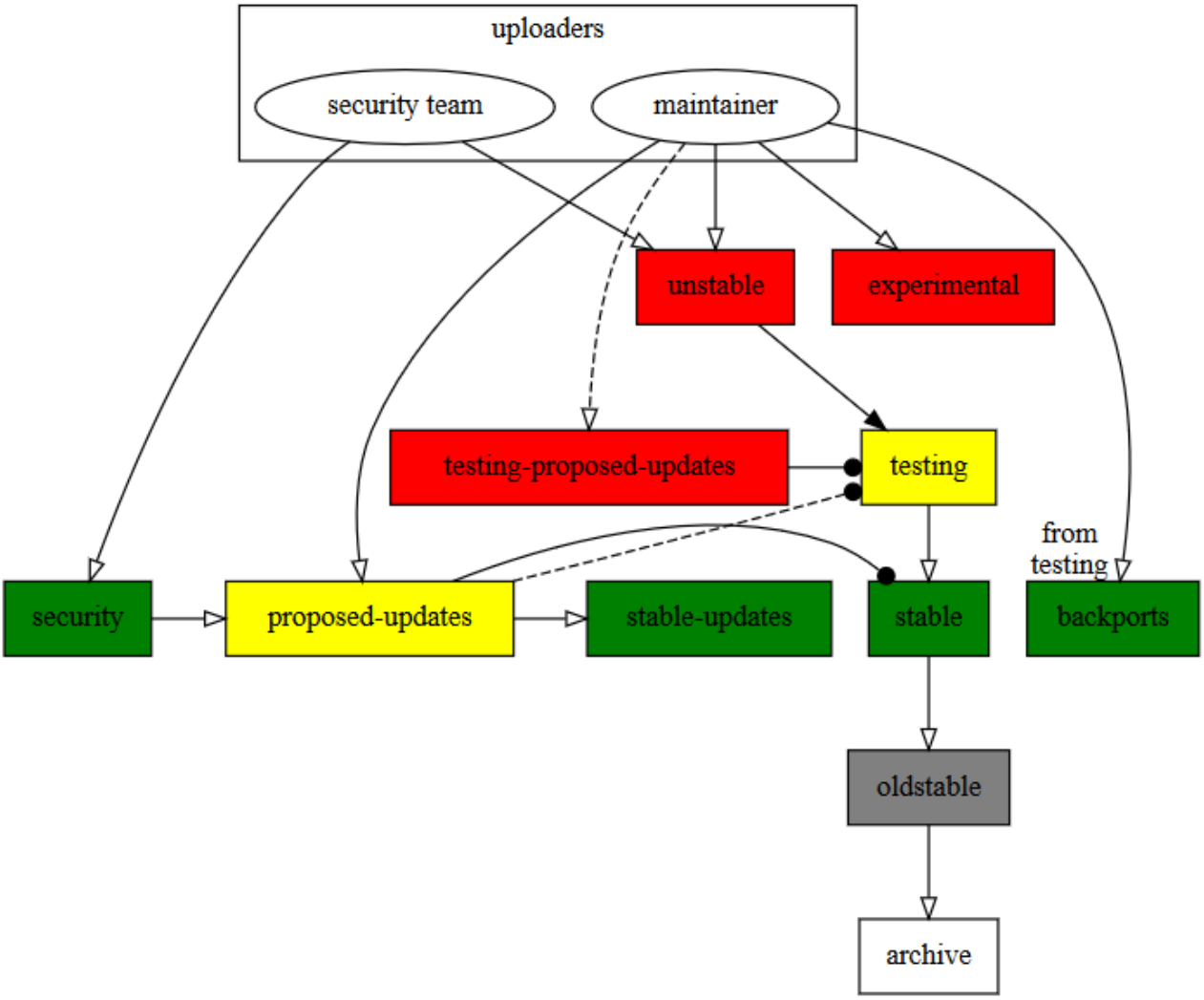


Fig. 3 Gestión de repositorios de la distribución de GNU/Linux Debian. Fuente (Debian 2018).

Todos los paquetes subidos por los desarrolladores de Debian son automáticamente almacenados en repositorio de la versión *unstable*, por lo que el único requerimiento necesario es que estos puedan ser construidos correctamente desde el código fuente (Yang 2017). No es recomendado utilizar en ambientes de producción esta versión (a menos que cuente con la experiencia necesaria para arreglar los errores o sea un probador o mantenedor de Debian) debido a las siguientes circunstancias:

- Es probable que el paquete que se desea instalar no tenga las dependencias cumplidas.
- Es probable que contenga serios errores y el equipo de seguridad de Debian no se responsabiliza por resolver ninguna vulnerabilidad en *sid*.

La distribución *testing* se genera de forma automática a partir de *unstable* gracias a un conjunto de *scripts*. Los paquetes que son seleccionados tienen poca probabilidad de tener errores críticos y

sus dependencias de ejecución y construcción estén disponibles en *sid* (Debian 2017a). Para que un paquete en *unstable* sea movido para *testing* debe cumplir con las siguientes condiciones:

- cuando ha estado de dos a diez días en *unstable*.
- debe estar compilado en todas las arquitecturas en las que ha sido compilado anteriormente en *unstable* y en la próxima versión.
- instalar el paquete en *testing* no hace a la distribución más inestable.
- el paquete no introduce errores críticos en la versión de la distribución.

Luego de un período de tiempo de desarrollo, una vez que el cronograma de lanzamientos lo demande, las políticas para la entrada de paquetes de *unstable* a *testing* se ajustan y los paquetes sensibles a errores se eliminan. En este período solamente son aceptados cambios para corrección de errores. Una vez solucionados todos los problemas existentes se realiza el lanzamiento de la nueva versión de Debian, por lo que la distribución *testing* es clonada y renombrada a *stable* (Barth, Di Carlo, Hertzog, Nussbaum, Schwarz, Jackson 2017).

Este proceso de gestión de versiones (y sus correspondientes repositorios) se basa en la suposición de que los paquetes almacenados en *unstable* pueden ser publicados en *stable* luego de un periodo de prueba y depuración en *testing*. A pesar de ser un proceso extenso y difícil de implementar, Debian posee un gran número de desarrolladores comunitarios que le permiten empaquetar prácticamente todo el software distribuido.

Dentro de los principales beneficios de los procesos de gestión de repositorios de la distribución de GNU/Linux Debian está el mecanismo para aprobar la entrada de paquetes al repositorio, así como el uso de varios repositorios dedicados a la realización de pruebas y para desarrollo.

1.3.2 **Proceso de gestión de los repositorios de Ubuntu.**

Debido a su filosofía, los repositorios de Ubuntu distinguen y separan todo el software brindado de acuerdo a su tipo de licencia. Por esta razón cada repositorio cuenta con cuatro componentes:

- main: software libre y de código abierto al cual Canonical[5] brinda soporte o desarrolla directamente.
- restricted: controladores propietarios para dispositivos.
- universe: software libre y de código abierto mantenido por la comunidad.
- multiverse: software restringido por *copyright* u otros asuntos legales.

[5] Canonical: compañía que se encarga del desarrollo, la venta del soporte técnico y servicios de Ubuntu y los productos afines.

Al ser liberada cada versión de Ubuntu, esta cuenta con un único repositorio (con nombre igual al nombre de código o *codename*), el cual es congelado con el objetivo de mantener la consistencia. A medida que nuevas actualizaciones van surgiendo o nuevos errores son corregidos, son creados los cuatro repositorios de actualizaciones de la distribución: *backports* para la inclusión de paquetes de distribuciones posteriores (Ubuntu 2016a), *proposed* para probar paquetes actualizados no liberados aún, *updates* para almacenar todas las actualizaciones y *security* para las actualizaciones de seguridad solamente (Waugh 2008).

El ciclo de vida de las actualizaciones de los paquetes en el repositorio de Ubuntu está regido por el procedimiento de Actualización de Lanzamiento Estable (SRU por sus siglas en inglés) y es aplicado solo bajo ciertas circunstancias como:

- Errores de alto impacto.
- Errores que solo afectan una aplicación y no un subsistema completo (como el kernel o el servidor gráfico).
- Adición de soporte para nuevo hardware.
- Nuevas funcionalidades que no afecten el comportamiento de las existentes.
- Actualizaciones de versiones menores de paquetes.
- Nuevas versiones de socios comerciales.

El procedimiento básico para la actualización de paquetes empieza por (Ubuntu 2016b):

1. Verificar que el error esté etiquetado como “Solución liberada” o cumpla las condiciones necesarias para ser incluido como actualización.
2. Verificar que el reporte de error sea público. En caso de ser privado serlo se realiza una copia pública del mismo.
3. Se actualiza la información del reporte de error.
4. Se sube el paquete fuente y la información complementaria para los servidores de Ubuntu, mediante el uso de la herramienta *dput*. Una vez subido el reporte de error se cambia a “En progreso” y una vez aceptado es automáticamente copiado al repositorio *proposed*.
5. El equipo del procedimiento de SRU revisa y acepta los cambios. Si todo está bien se mueve el paquete desde el repositorio *proposed* al de *updates*, luego de pasado el tiempo de prueba (mínimo siete días).

El repositorio *security* de Ubuntu es usado solamente para almacenar los paquetes actualizados debido a problemas de seguridad de los componentes *main* y *restricted* del repositorio principal. Normalmente todos los paquetes copiados a este repositorio son copiados para el repositorio *updates* también (Ubuntu Security Team 2017).

1.3.3 **Proceso de gestión de repositorios de Nova.**

Para el realizar el estudio de la gestión de repositorios de Nova es obligatorio mencionar el intento de desarrollar una herramienta de gestión y construcción de repositorios que, aunque nunca llegó a contener todas las funcionalidades deseadas, si brindó muchas experiencias en los sistemas construcción de paquetes y de gestión de repositorios.

1.3.3.1 **Nova distributed build system (ndbs).**

Durante el periodo 2011-2013 se creó un proyecto en Nova para el desarrollo de un sistema de creación y compilación distribuida de sus repositorios llamado NDBS (*Nova distributed build system*), el cual pretendía ser en una aplicación cliente-servidor que resolviera los problemas de la construcción de un repositorio de software para Nova desde cero (Pérez Herrera 2013). Dentro de sus principales funcionalidades estaban la gestión de las llaves criptográficas para la firma digital de paquetes y repositorios, un sistema de gestión de repositorios mediante un servicio web externo, la gestión de los nodos de compilación y la compilación distribuida de los paquetes estableciendo una estrategia de construcción en la cual se priorizaban aquellos con mayor número de dependencias reversas (los paquetes de los que más se dependen) (Rabelo, González, Pérez, Delgado 2015).

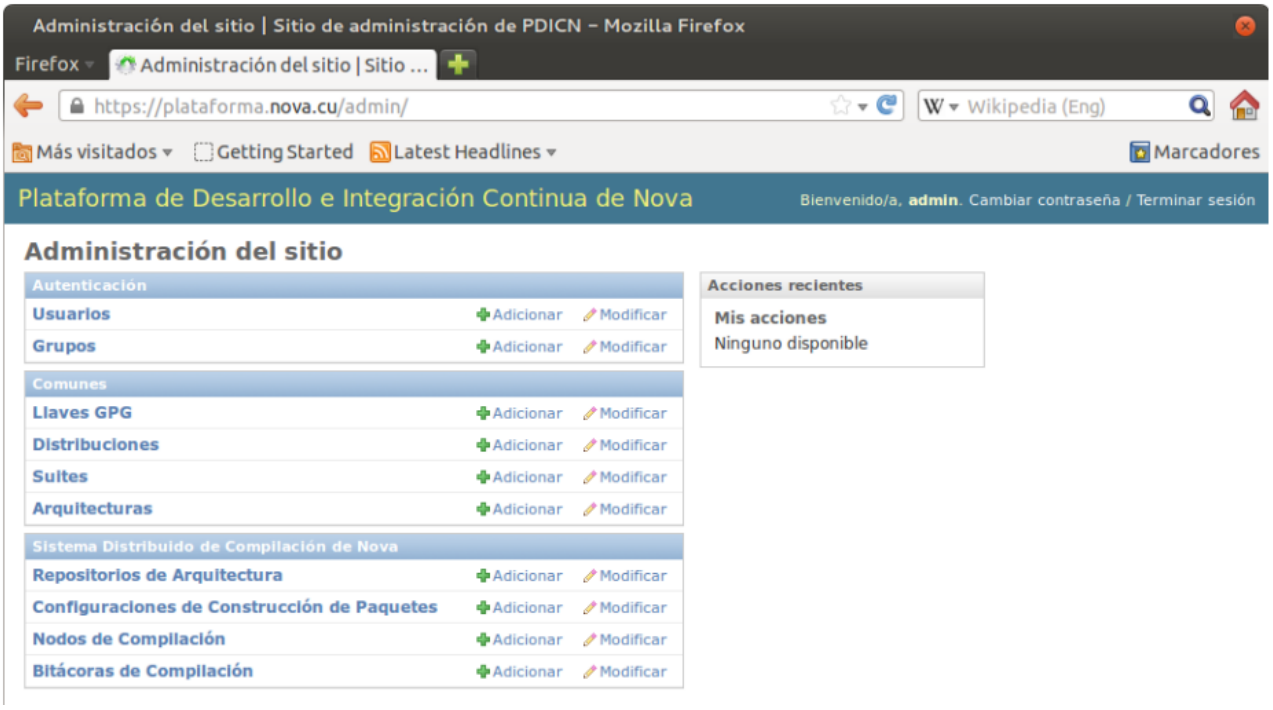


Fig. 4 Plataforma de compilación distribuida de Nova NDBS. Fuente (Pérez Herrera 2013).

Este proyecto no alcanzó los resultados esperados debido a la inestabilidad y gestión deficiente causando que muchas de las funcionalidades descritas tuvieran errores (como el caso de la gestión de las firmas digitales), y otras nunca llegaran a ser desarrolladas completamente como el sistema

de gestión de repositorios, resultando solamente en una herramienta para construir paquetes. Con el paso del tiempo y cuando no quedaba nadie del personal original del proyecto, esta herramienta fue descartada debido a que era más costoso darle mantenimiento y arreglar los errores surgidos que usar una herramienta de automatización libre ya existente.

1.3.3.2 **Proceso actual de gestión de repositorios de paquetes en Nova.**

Para la gestión de actualizaciones de los repositorios de Nova, los administradores de la configuración adicionan la versión de la distribución al fichero *basedir/conf/distributions* y las fuentes de actualizaciones en el fichero *basedir/conf/updates*. Luego se verifica que una actualización del repositorio no actualice algún paquete modificado por los desarrolladores de Nova mediante la opción *check-pull* de reprepro y se actualizan los que no contienen ningún conflicto. En caso de actualización de paquetes de Nova, se genera una lista con dichos paquetes que es entregada a los desarrolladores para que procedan a reaplicar las modificaciones de Nova sobre los nuevos paquetes. Estas modificaciones son subidas a los servidores y procesadas mediante el mecanismo de entrada.

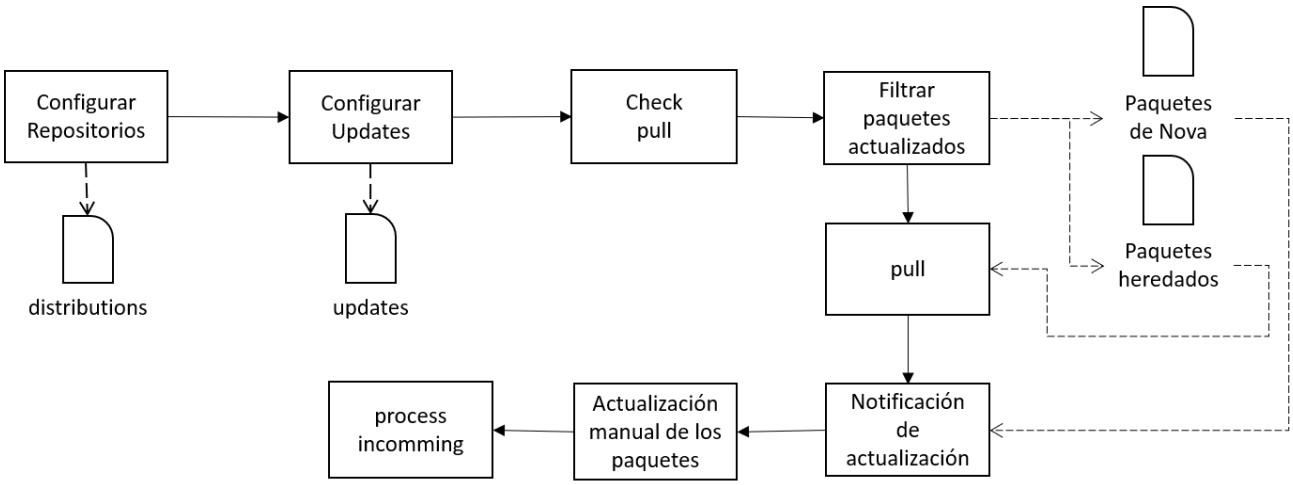


Fig. 5 Proceso de actualización de los repositorios de Nova.

Este proceso de trabajo presenta las siguientes deficiencias:

- Se realiza de forma manual y no existe automatización en el proceso.
- Entrada de actualizaciones binarias sin ser antes construidas desde el código fuente.
- Los paquetes heredados y los paquetes modificados o desarrollados por Nova son almacenados en la misma estructura de los repositorios. Esto puede dar al traste a que se sobrescriban los paquetes de Nova y se pierda el trabajo.
- Una vez completada el proceso es imposible devolver el repositorio a estados previos en caso de error.
- Son heredados todos los problemas de reprepro.

1.4 Herramientas de construcción y gestión de repositorios de software en Debian.

Son múltiples las herramientas para la creación de repositorios de Debian. Muchas no brindan las funcionalidades básicas como la creación de repositorios personalizados, aceptable rendimiento con grandes repositorios, capacidad de importar réplicas de repositorios o poca integración con sistemas de firmas digitales. Debido a esta razón solamente se estudian las herramientas dak, reprepro y aptly.

1.4.1 Debian Archive Kit (Dak).

La herramienta dak consiste en una serie de scripts que automatizan la generación de repositorios en Debian y es parte de un largo ecosistema de software que realiza el proceso de compilación y publicación de paquetes en Debian (Debian 2017b). Esta herramienta está desarrollada en los lenguajes de programación *python*, *bash* y *perl* y utiliza como gestor de base de datos a *postgresql*.

Todo el proceso de trabajo con dak se basa en el mecanismo de entrada de paquetes y las políticas de Debian para aceptarlos o denegarlos. El mecanismo de entrada es el responsable de coleccionar las actualizaciones de los paquetes e instalarlos en el repositorio de Debian y consiste en un conjunto de directorios y scripts instalados en http://ftp-master.debian.org (Barth, Di Carlo, Hertzog, Nussbaum, Schwarz, Jackson 2017).

Los paquetes subidos por los mantenedores, haciendo uso de las herramientas *dupload* y *dput*, son almacenados en un directorio llamado *UploadQueue*. Este directorio es verificado cada cierto tiempo buscando por los archivos **.changes* correctamente firmados, los que son movidos junto con sus correspondientes archivos para el directorio *unchecked*. Este último es verificado cada 15 minutos por un script llamado *dakprocess-upload*, el cual verifica la integridad de los archivos y su firma criptográfica. Si los paquetes están listos para ser instalados son movidos para el directorio *done* para ser archivados en el repositorio, pero si es un nuevo paquete (o el paquete fuente genera algún paquete binario nuevo) es movido para el directorio *new* donde esperará a que los administradores lo aprueben. Por otra parte, si el paquete necesita algún tipo de configuración manual es movido para el directorio *byhand* donde esperará a que los administradores lo instalen manualmente. Si en alguno de los casos previos hubo un error el paquete será movido para el directorio *rejected* (Debian 2013).

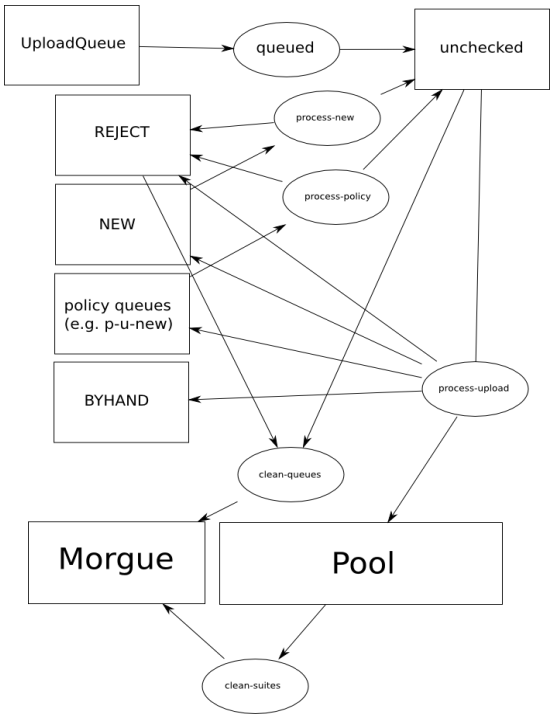


Fig. 6 Mecanismo de entrada en los repositorios de Debian con dak. Fuente (Debian 2013).

Al ser la herramienta oficial de Debian, está diseñada para la gestión de grandes repositorios, con varias versiones y múltiples componentes, y brinda la posibilidad de firmar digitalmente los repositorios utilizando GPG. A pesar de la importancia de su rol dentro del ambiente de trabajo de Debian, dak carece de documentación para su despliegue y desarrollo por lo que resulta extremadamente difícil utilizarlo fuera de los servidores de Debian. Debido a la política de desarrollo de la distribución, de que todos los paquetes deben ser construidos en el código fuente, dak no permite la inclusión de paquetes binarios previamente generados sin sus paquetes fuentes al repositorio. Aunque muchas de las operaciones de dak son completamente automatizadas, esta herramienta requiere una gran cantidad de vigilancia, debido a muchas de las acciones son moderadas por los administradores del repositorio de Debian (Barth, Di Carlo, Hertzog, Nussbaum, Schwarz, Jackson 2017).

1.4.2 **Reprepro.**

Reprepro (formalmente conocido como *mirrorer*) es una herramienta para gestionar repositorios de paquetes de Debian (binarios o de código fuente), capaz de instalar los paquetes en su debido lugar y generar los índices necesarios, para que los gestores de paquetes de las distribuciones basadas en Debian, como *apt*, puedan utilizar los repositorios gestionados. Entre sus funcionalidades se encuentra la capacidad de crear repositorios espejos parciales de repositorios remotos, incluyendo la fusión de distintas fuentes, así como la eliminación automática de los paquetes no disponibles en la fuente original (Link 2011).

Esta herramienta permite la creación de diferentes repositorios mediante los archivos de configuración ubicados en *basedir/conf/distributions* en el cual se especifican nombre de código del repositorio (*codename*), los componentes (*components*), las arquitecturas soportadas, así como otros campos de información del repositorio. Para la inclusión de paquetes individuales, reprepro brinda la utilidad *includedeb* especificándole el *codename* destino y la lista de paquetes. Además, pueden ser incluidos los archivos **.changes* resultado de la creación de paquetes binarios y código fuente, mediante la opción *include*. Reprepro brinda una amplia gama de opciones que permiten las operaciones básicas con los diferentes repositorios como listar, copiar, mover o eliminar paquetes.

Reprepro permite la creación de espejos parciales de repositorios remotos definidos en el archivo de configuración *basedir/conf/updates*, haciendo uso de los mismos métodos que utiliza *apt* (*/usr/lib/apt/methods/*). Es posible importar los paquetes de los diferentes espejos como fuente de actualización de los repositorios configurados mediante la especificación de los nombres de los espejos en el archivo de los repositorios.

Reprepro brinda un mecanismo de entrada para la subida de paquetes como resultado de la construcción de estos por parte de algún mecanismo de compilación. Este mecanismo puede ser configurado a través del archivo de configuración *basedir/conf/incomming*, en el cual deben ser especificado el nombre de la cola de entrada, la dirección del directorio en que se van a almacenar los paquetes, una dirección temporal donde se verifican las sumas de verificación de los paquetes y los permisos, la lista de distribuciones en las cuales van a ser incluidos los paquetes y los permisos de los desarrolladores para incluir los paquetes. Esta herramienta permite la creación de múltiples colas para el mecanismo de entrada (Link 2013).

A pesar de ser la utilizada en Nova desde el año 2011, reprepro cuenta con múltiples problemas de diseño. El más significativo es el uso de bases de datos Berkeley DB, lo cual, a decir de sus propios desarrolladores, fue un gran error debido a la posibilidad de corrupción ante insuficiente espacio de disco o interrupción (Link 2015). Otro problema es que almacena los paquetes directa y solamente en la carpeta pool del repositorio, directorio que va siendo construido a medida que se les adicionan los paquetes a los repositorios. Este enfoque aumenta la rapidez a la hora de realizar operaciones en la carpeta *pool* del repositorio, pero con ella se corre el riesgo de que, ante una acción errónea, puedan ser borrados los paquetes de la distribución. A pesar de la existencia de una opción para hacer instantáneas en el repositorio, estas no pueden ser publicadas de la forma oficial que recomiendan los estándares de los repositorios de Debian y no existe alguna opción para eliminarlos (solamente manual), por lo que su uso no es extendido y está pobremente documentado (Link 2011).

1.4.3 **Aptly**

Aptly es una herramienta que pretende ser la navaja suiza de la gestión de repositorios en Debian, brindando múltiples funcionalidades como la creación de espejos de repositorios remotos, administrar repositorios personales locales, tomar instantáneas (de repositorios locales y espejos de repositorios remotos), filtrar paquetes con sus dependencias y publicarlas (Smirnov 2016). El principal objetivo de aptly es establecer repetibilidad y control sobre los cambios de los paquetes del repositorio, produciendo un conjunto fijo de paquetes. Al mismo tiempo, es capaz de realizar cambios controlados y precisos sobre el contenido del repositorio, así como la transición del paquete a una nueva versión o revertirlo a una versión anterior.

El funcionamiento de aptly se base en las siguientes entidades centrales (Smirnov 2013):

- *Mirror* (Réplica): consiste en una réplica de los índices y la lista de paquetes de un repositorio remoto. Es posible realizar la réplica aplicando filtros basado en las arquitecturas o información de los paquetes (réplica parcial).
- Repositorio local: consiste en un repositorio local al cual se le pueden adicionar o eliminar paquetes de forma sencilla.
- *Snapshot* (Instantánea): constituye una lista inmutable de paquetes generada a partir de un repositorio o una réplica. Es posible realizar varias operaciones sobre estos para obtener otros *snapshots* como verificaciones de dependencias, filtrarlos, representar las diferencias entre dos *snapshots* y unir múltiples *snapshots*.
- Repositorio publicado: representación publicada de un *snapshot* generado o de un repositorio local, listo para ser utilizado por los gestores de paquetes como *apt*.

A diferencia del resto de las herramientas de gestión de paquetes, aptly no se vale de la estructura estandarizada de un repositorio de Debian (las carpetas *dist* y *pool*) para sus operaciones internas. Aptly utiliza una base de datos local (almacenada en la carpeta *db*) para almacenar la información de los paquetes y una carpeta (llamada *pool*) donde almacenar, bajo una estructura propia definida, todos los paquetes que son gestionados por alguna de las entidades.

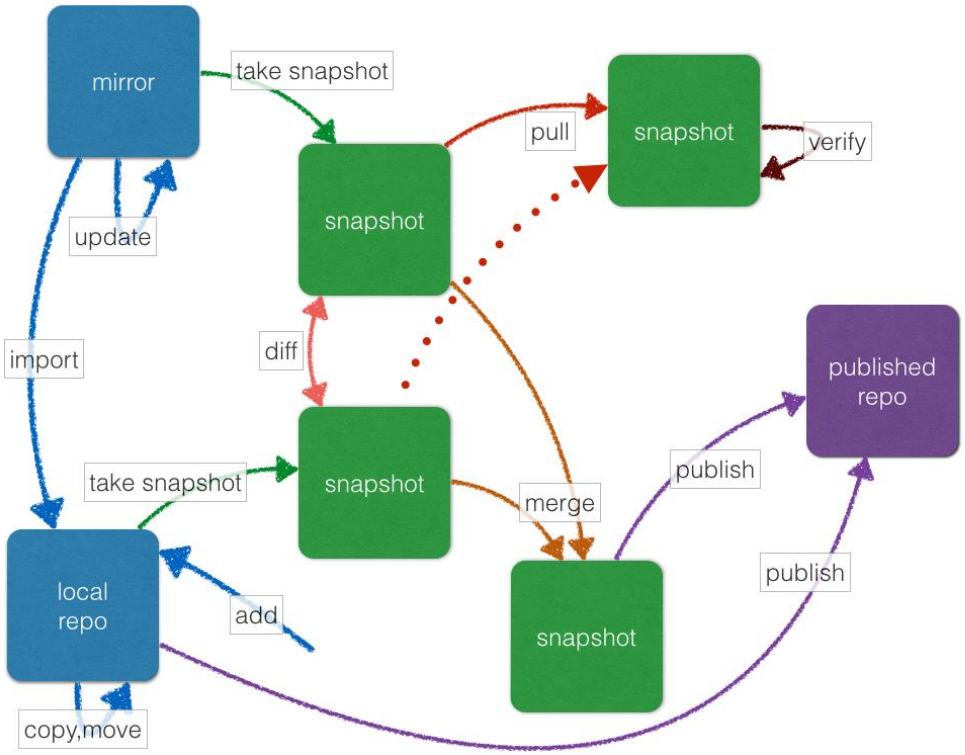


Fig. 7 Transición entre entidades en aptly. Fuente (Smirnov 2013).

El flujo de trabajo propuesto por aptly comienza con la creación de réplicas de repositorios remotos y repositorios locales. Los repositorios locales pueden ser modificados mediante la adición e inclusión de paquetes manualmente, importándolos de una réplica creada o copiándolos y moviéndolos entre repositorios locales. En cualquier momento puede crearse instantáneas de los repositorios y de las réplicas y aplicársele cualquiera de las operaciones definidas por aptly. A su vez estas instantáneas pueden ser publicadas, generándose o actualizándose la estructura estandariza de un repositorio Debian (Smirnov 2013).

Aptly se integra con *GNU Privacy Guard* (*GnuPG* o *gpg*)[6] desde la primera versión y lo utiliza para la validación de las firmas de los paquetes y para firmar los repositorios publicados. A partir de la versión 1.1.0, aptly incluye la funcionalidad para administrar varios proveedores de validación y firma, e incluye un nuevo proveedor basado en la implementación nativa de *OpenPGP* del lenguaje *Go* (lenguaje en el que está desarrollado aptly), permitiendo cambiar de proveedor a demanda.

Otras funcionalidades de aptly que facilitan el trabajo de los administradores de los repositorios son (Smirnov 2016):

[6] GnuPG o GPG es una completa y libre implementación del estándar OpenPGP definido por RFC4880, también conocido por PGP (GnuPG 2018).

- Configuración del destino final de la publicación de repositorios: aptly provee tres vías para ubicar la publicación de los repositorios: publicación en el sistema de archivos, publicación en la plataforma Amazon S3 y la publicación en OpenStack Swift.
- Publicación de múltiples versiones de un mismo paquete en un mismo repositorio.
- Resolución automática de dependencias de paquetes.
- Mecanismo de entrada para los repositorios.
- API para la gestión remota y concurrente de los repositorios.
- Verificación de dependencias entre paquetes insatisfechas en un *snapshot*.

1.5 Conclusiones parciales

1. A partir de la revisión de la bibliografía en el área de repositorios de GNU/Linux se realizó un estudio de los estándares definidos por Debian que rigen los lineamientos de seguridad y estructura de los mismos.
2. Se realizó el estudio de la métodos y procedimientos para la gestión de repositorios de las distribuciones Debian, Ubuntu y Nova, evidenciándose las carencias de Nova en esta área que pudieran dar al traste con violaciones de seguridad y pérdida del trabajo realizado.
3. Se realizó un estudio de las herramientas para la gestión de los repositorios de las distribuciones basadas en Debian, así como sus deficiencias y potencialidades.

Capítulo 2. Procedimiento de gestión de repositorios de paquetes de Nova.

En este capítulo se realiza la evaluación de las herramientas de gestión de repositorios de software y se presenta el procedimiento propuesto para la gestión de los repositorios de paquetes de software de la distribución cubana de GNU/Linux Nova. Además, se describen, fundamentan y explican sus características generales y sus principales componentes, así mismo se realiza el diseño ingenieril de su implementación.

2.1 Evaluación de las herramientas de gestión de repositorios de paquetes de distribuciones basadas en Debian.

Para realizar la evaluación de las herramientas de gestión de repositorios de paquetes de Debian es necesario utilizar alguna estrategia existente para la evaluación y selección de software, específicamente software libre. QSOS (Qualification and Selection of Opensource Software) es un proyecto que brinda métodos y herramientas para calificar, seleccionar y comparar componentes de software libre, permitiendo automatizar y mutualizar la vigilancia tecnológica (QSOS 2017).

El proceso definido por QSOS está constituido por cuatro elementos:

1. Definir los criterios de evaluación del software
2. Evaluación del software: se evalúan los criterios seleccionados en el paso uno con una puntuación que va desde 0 (funcionalidad no cubierta) hasta 2 (funcionalidad cubierta completamente).
3. Calificar: se definen un conjunto de elementos que traduzcan las necesidades y las limitaciones relacionadas con el enfoque de selección de una pieza de software de código abierto. Puede realizarse mediante la aplicación de filtros de identidad, de madurez y cubrimiento funcional.
4. Selección del software que mejor cubra las necesidades del usuario, basándose en los resultados de los pasos anteriores.

Para seleccionar los criterios de las herramientas de gestión de repositorios a evaluar se utiliza el método de Análisis de la Prospectiva Estratégica que permite describir un conjunto de enfoques y tendencias para mejorar la toma de decisiones, implicando reflexionar acerca de las oportunidades y retos emergentes (Vasco 2011).

Para la aplicación de este método se efectúa el análisis estructural del sistema, el cual consiste en una herramienta de estructuración de una reflexión colectiva y ofrece la posibilidad de describir un sistema mediante el relevamiento de las variables esenciales para la evolución del mismo, apoyándose en una matriz que relaciona todos sus elementos consecutivos (Godet, Monti, Meunier, Roubelat 2000). Para su realización es necesario conformar un grupo focal compuesto por expertos en la gestión de repositorios de software (Anexo 1: Lista de expertos y actores en la gestión de repositorios de software libre.) y es factible auxiliarse en el software MicMac.

El análisis estructural contiene las siguientes fases:

1. Listado de las variables que caracterizan el sistema: son enumeradas las variables que caracterizan a las herramientas de gestión de repositorios mediante la realización de talleres de prospectiva. El grupo de expertos extrajo las siguientes variables claves (Anexo 2: Lista de variables claves del análisis estructural.):
    - Documentación.
    - Facilidad de uso.
    - Gestión de grandes repositorios.
    - Integración con sistemas de firma digital.
    - Recuperabilidad ante errores.
    - Réplicas de repositorios remotos.
    - Sistema de verificación de integridad y dependencias.
    - Mecanismo de entrada de paquetes.
    - Inclusión de paquetes sin código fuente.
    - Mecanismo de gestión de instantáneas.
    - Integración con servicios externos de integración continua.
    - Posibilidad de inclusión de múltiples versiones de un mismo paquete.
2. Descripción de las relaciones entre las variables: son definidas las relaciones entre las variables extraídas de forma cualitativa mediante una matriz de relaciones directas y utilizando la siguiente notación:
    - 0: no hay influencia.
    - 1: influencia débil.
    - 2: influencia mediana.
    - 3: influencia fuerte.
    - 4 influencia potencial.

Dando como resultado la siguiente matriz:

	1 : Doc	2 : Fac	3 : Big	4 : PKI	5 : Fix	6 : Mir	7 : Ver	8 : Inc	9 : Cod	10 : Sna	11 : CIS	12 : Mul
1 : Documentación	0	P	3	P	3	2	3	2	0	2	P	2
2 : Facilidad de uso	0	0	2	2	2	0	0	2	0	1	2	1
3 : Gestión de grandes repositorios	2	1	0	3	P	P	P	2	1	P	2	2
4 : Integración con sistemas de firma digital.	1	2	P	0	0	0	2	3	0	0	1	0
5 : Recuperabilidad ante errores	3	2	3	1	0	1	2	0	0	P	0	0
6 : Réplicas de repositorios remotos	0	0	P	2	0	0	3	0	1	2	0	3
7 : Sistema de verificación de integridad y dependencias	2	2	P	3	3	1	0	2	0	1	0	0
8 : Mecanismo de entrada de paquetes	0	1	P	3	0	0	2	0	0	2	P	2
9 : Inclusión de paquetes sin código fuente	0	0	2	0	0	0	1	0	0	0	0	2
10 : Mecanismo de gestión de instantáneas	3	3	P	2	P	3	2	1	0	0	2	1
11 : Integración con servicios externos de integración continua	3	2	2	1	0	0	0	P	0	2	0	1
12 : Posibilidad de inclusión de múltiples versiones de un mismo paquete	0	0	2	0	0	0	0	P	2	2	2	0

Fig. 8 Realización de la Matriz de influecia directa

3. Identificación de las variables claves: son detectadas las variables esenciales para la evolución del sistema, primero mediante la matriz de influencia directa (MDI) y luego a través la matriz de influencia indirecta (MII). Mediante el uso de MicMac, se generan los planos de motricidad (mide la acción que realiza la variable en el sistema) y dependencia (mide cómo reacciona este factor a los cambios en el sistema) para ambas matrices. En el siguiente gráfico se muestran los valores calculados de las variables en el plano de influencia/dependencia para MII.

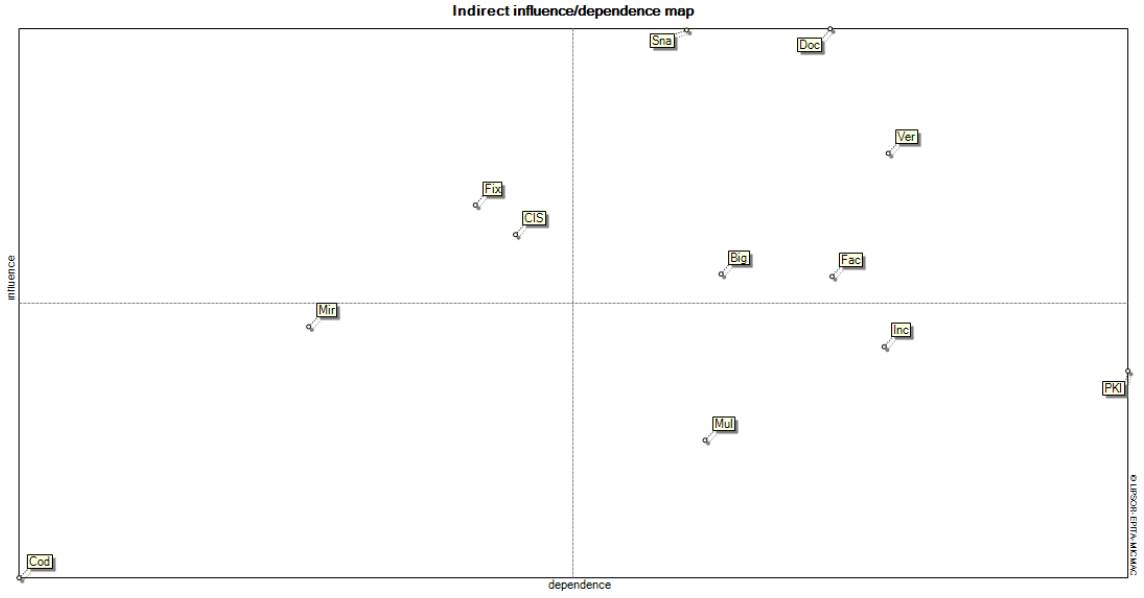


Fig. 9 Plano de influencia/dependencia indirecta del sistema

Una vez calculados los valores de influencia/dependencia de las variables en el plano se procede a su clasificación según el cuadrante que les corresponda:

Variables excluidas: son variables poco motrices y poco dependientes que por lo general corresponden a tendencias pasadas del sistema o están desconectados de él, por lo que no constituyen parte determinante del sistema. En esta investigación solo existe una variable con esta clasificación:

- Inclusión de paquetes sin código fuente.

Variables del pelotón: son variables que se encuentran cercanas al borde del primer cuadrante del plano por lo que tienen un nivel medio de motricidad y/o dependencia. Por lo general corresponden a funcionalidades de valor añadido al sistema (mediana influencia y poca dependencia) o determinan el buen funcionamiento del sistema (baja/mediana influencia y mediana dependencia). En esta investigación solo existe una variable con esta clasificación:

- Réplicas de repositorios remotos.

Variables de entrada o determinantes: poseen un alto nivel de motricidad y poca dependencia en el sistema y pueden convertirse en frenos o motores del sistema en dependencia de su evolución. En esta investigación existen dos variables con esta clasificación:

- Recuperabilidad ante errores.
- Integración con servicios externos de integración continua.

Variables enlace: se encuentran en el cuadrante superior derecho del plano y tienen un alto valor de motricidad y dependencia. Por lo general son de naturaleza inestable y se corresponden a los retos del sistema. En esta investigación existen cinco variables con esta clasificación:

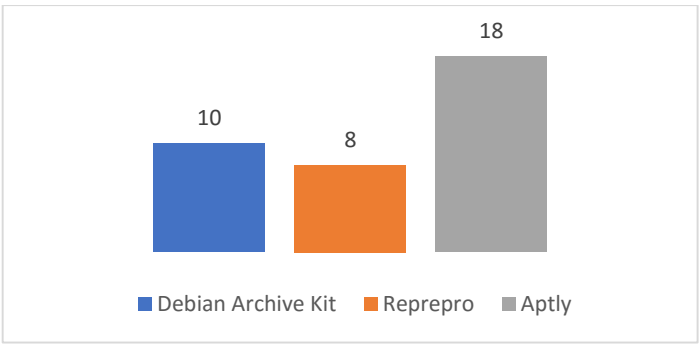
- Mecanismo de gestión de instantáneas.
- Documentación.
- Sistema de verificación de integridad y dependencias.
- Gestión de grandes repositorios.
- Facilidad de uso.

Variables resultado: se caracterizan por su baja motricidad y su alta dependencia por lo que están situadas en el cuadrante inferior derecho y constituyen un indicador descriptivo de la evolución del sistema. En esta investigación existen cinco variables con esta clasificación:

- Posibilidad de inclusión de múltiples versiones de un mismo paquete.
- Mecanismo de entrada de paquetes.
- Integración con sistemas de firma digital.

La aplicación del análisis estructural da como resultado la necesidad de evaluar once de las variables, excluyendo solamente la variable Inclusión de paquetes sin código fuente, debido a la poca influencia en el sistema.

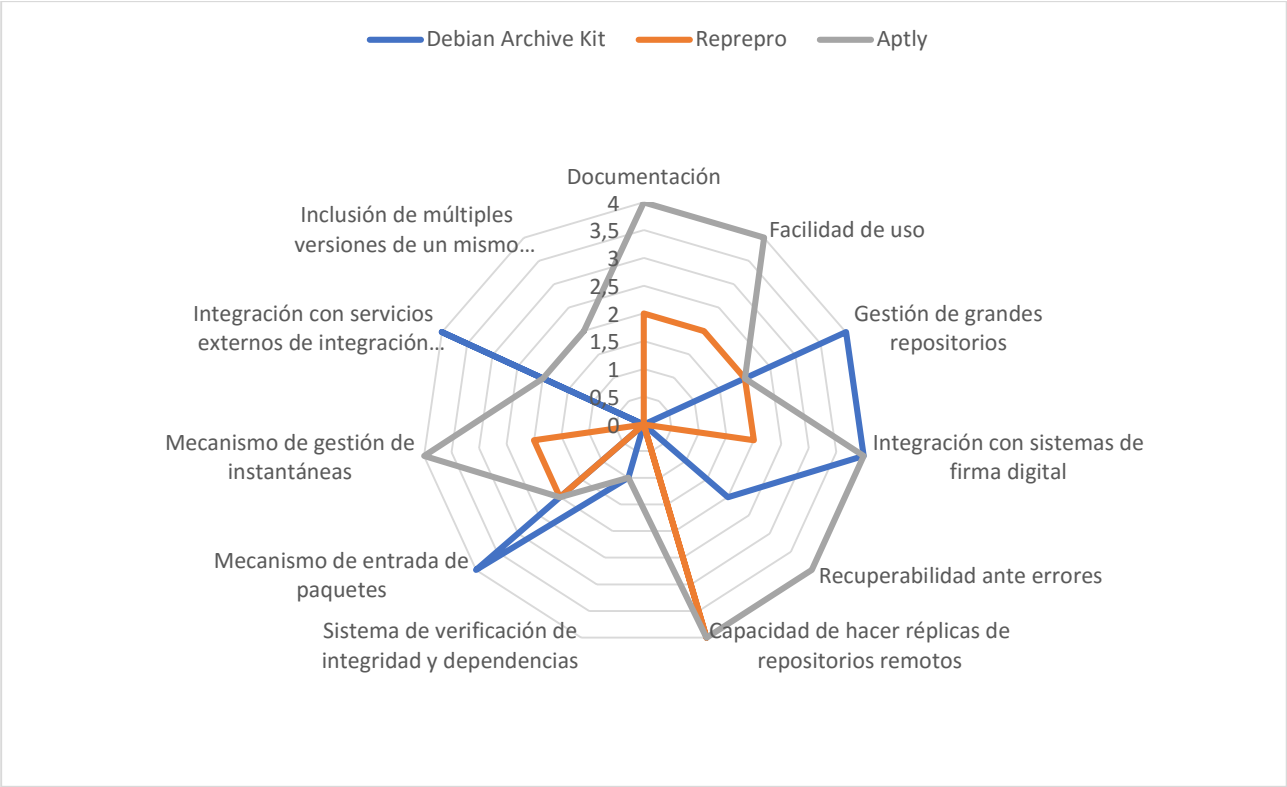
Continuado con QSOS, y luego de definidos los criterios de evaluación se pasa a la evaluación de estos en cada una de las herramientas seleccionadas (ver Anexo 1: Lista de expertos y actores en la gestión de repositorios de software libre.) dando como resultado:



Gráfica 1 Evaluación de los criterios para la selección de la herramienta para la gestión de repositorios de paquetes de Debian.

A continuación, se procede a calificar la evaluación realizada en el paso anterior. Para esto se definen los siguientes filtros:

- Compatible con Nova: el software está presente en los repositorios o es compatible con la distribución de GNU/Linux Nova. Todas las herramientas utilizadas son compatibles con Nova.
- Cubrimiento funcional: aplicado a los criterios de evaluación y cuantificado en funcionalidad requerida (2), funcionalidad opcional (1) y funcionalidad no requerida (0) (ver Anexo 4: Cubrimiento funcional de los criterios de evaluación.)



Gráfica 2 Evaluación de los criterios de selección de las herramientas de gestión de repositorios de paquetes de Debian.

Para obtener la evaluación de las herramientas de gestión de repositorio se multiplica la evaluación de los criterios seleccionado con el cubrimiento funcional de los criterios (ver Anexo 5: Evaluación de las herramientas de gestión de repositorios aplicando QSOS.). El resultado obtenido es expuesto en Gráfica 2 para su mejor entendimiento

El resultado de aplicar el proceso definido por QSOS brinda que la herramienta con mayor adecuación funcional de las estudiadas, según los criterios de selección definidos, es aptly.

2.2 Características de los repositorios de Nova.

El repositorio de software de Nova está compuesto por paquetes heredados de los repositorios de la distribución Ubuntu (de las versiones con soporte de largo término o versiones LTS), así como otros repositorios oficiales compatibles con esta. Debido a esta razón la mayoría del software brindado solo es recompilado y añadido a los repositorios, a excepción de una minoría que son creados o modificados por el equipo de desarrollo de Nova u otras asociaciones y comunidades nacionales para adaptar el software a las necesidades y realidades tecnológicas y sociales de Cuba.

El código fuente de todos los paquetes modificados o creados por Nova están almacenados en un servidor de control de versiones GIT, desde donde son descargados y procesados para construir sus respectivos paquetes de código fuente y sus correspondientes binarios.

2.3 Descripción general del proceso.

El proceso propuesto para la gestión de los repositorios de software de Nova empieza por la definición de los repositorios de software heredado y los de software propio a ser utilizados en la distribución cubana de GNU/Linux. Basándose en esta información, se descargan las réplicas de software heredado, se hallan las diferencias con los paquetes de desarrollo propio y se publican, generándose la estructura del repositorio lista para ser usada luego por los gestores de paquetes.

En el procedimiento propuesto se opera sobre listas de paquetes invariables llamadas instantáneas, a diferencia de los procesos estudiados de otras distribuciones y los precedentes de Nova, que se centran en gestionar los ficheros en la estructura física del repositorio.

Como resultado de la ejecución del procedimiento es obtenido un repositorio de la distribución con tres componentes: principal con todo el software con soporte oficial de la distribución, extendido con el resto del software de Nova y un componente de actualizaciones con los paquetes fuentes de software heredado que actualizan (o que no están presentes) paquetes del repositorio. Este último es solo para trabajo interno del proyecto y es útil para proveer un único lugar desde donde los desarrolladores puedan descargar el código fuente para luego modificarlo, construirlo y luego subirlo al repositorio como parte del software de Nova. Además, son creados un conjunto de reportes útiles para conocer la integridad del repositorio publicado, que paquetes contienen dependencias (de construcción o ejecución) fallidas y que paquetes propios de Nova van a ser actualizados.

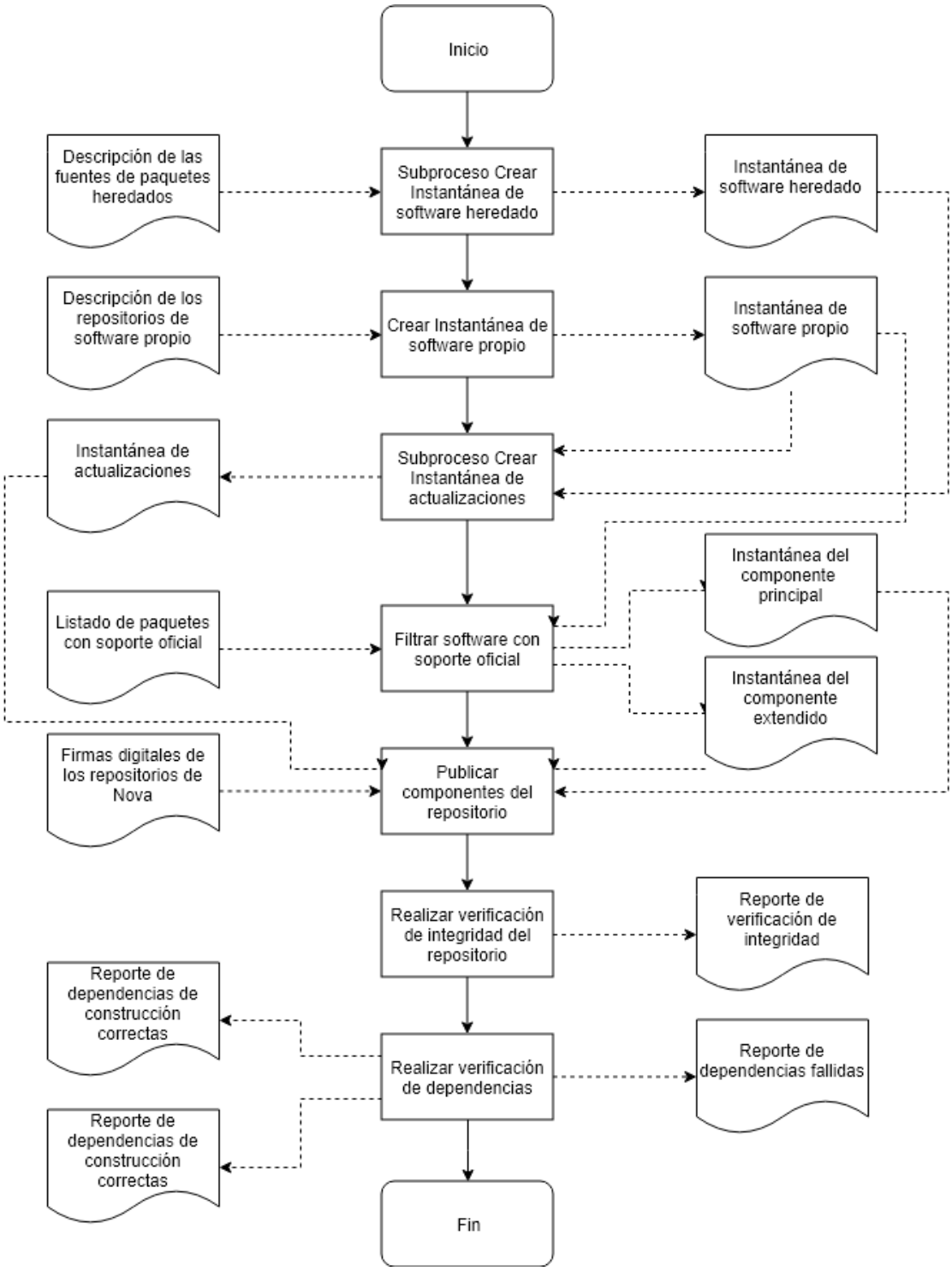


Fig. 10 Propuesta de procedimiento para la gestión de repositorios de Nova.

2.3.1 Indicadores del estado del repositorio recolectados en el procedimiento.

Durante la ejecución del procedimiento se recolectan los siguientes indicadores a través de los reportes generados en los diferentes pasos:

1. Dependencias de dependencias de instalación fallidas: cantidad de paquetes en el repositorio que no pueden ser instalados debido a que no existe en el repositorio alguna de sus dependencias.
2. Dependencias de dependencias de construcción fallidas: cantidad de paquetes de código fuente que no pueden ser construidos utilizando los paquetes del repositorio como fuente de dependencias de construcción.
3. Integridad de los índices: cantidad de índices del repositorio que no poseen la suma de verificación declarada en el archivo *Release*.
4. Integridad de los paquetes binarios: cantidad de paquetes binarios que no tienen la suma de verificación declarada en el archivo *Packages*.
5. Integridad de los paquetes de código fuente: cantidad de paquetes fuentes que no tienen la suma de verificación declarada en el archivo *Sources*.
6. Autenticidad de los índices: cantidad de índices no firmados con la llave del archivo oficial de Nova.

Estos indicadores son útiles para determinar la confiabilidad, la integridad y la completitud del repositorio y, junto con los reportes generados, brindan información útil para la realización de acciones que aumentar la calidad general del repositorio.

2.4 Componentes del procedimiento propuesto.

El procedimiento propuesto está compuesto por los siguientes elementos:

- Subproceso Crear instantánea de software heredado: basándose en la información de la lista de los repositorios de software heredado, se crean las réplicas, se actualizan, se crea una instantánea por cada una y se unen, formando la instantánea de software heredado.
- Crear Instantánea de software propio: crea una instantánea del repositorio de software propio basándose en la información de este.
- Subproceso Crear Instantánea de actualizaciones: crea un reporte de los paquetes que se van a actualizar, así como una instantánea con los paquetes fuentes de las actualizaciones.
- Filtrar software con soporte oficial: filtra la instantánea del repositorio de software de Nova, creando una instantánea del componente principal con todos los paquetes con soporte oficial (y sus dependencias) y la instantánea del componente extendido con el resto.
- Publicar ramas del repositorio: son publicadas todos los componentes (principal, extendido y de actualizaciones) del repositorio, siendo firmados los índices con la llave digital de Nova.

- Realizar verificación de integridad del repositorio: es verificado que todos los paquetes de los índices estén presentes en el repositorio y que todos los metadatos declarados en el archivo *Release* estén presentes. Además, se comprueba la firma de este último.
- Realizar verificación de dependencias: verifica que todos los paquetes en el repositorio tengan las dependencias de ejecución cumplidas (o construcción en caso de ser paquetes fuentes) y crea un informe con la información recolectada.

A continuación, se detallan cada uno de estos elementos, describiendo las acciones tomadas en cada uno, sus entradas y sus salidas.

2.4.1 **Subproceso Crear instantánea de software heredado.**

El subproceso para crear la instantánea de software heredado permite descargar todo el software reutilizado en Nova de otras distribuciones y crear una instantánea con este. El proceso empieza por la creación de las réplicas de los repositorios de software heredado especificadas en el documento Descripción de fuentes de paquetes heredados (ver Tabla 1). Una vez creadas se procede a actualizarlas, proceso en el que son descargados todos los paquetes de las réplicas.

Una vez creadas y actualizadas las réplicas de software heredado, se procede a crear una instantánea por cada réplica, para luego fusionarlas todas en una única instantánea que contenga todo el software heredado. En este proceso de fusión de instantáneas se unen todas las listas de paquetes de las réplicas en una sola, en caso de que se encuentre múltiples paquetes con el mismo nombre y arquitectura se favorece siempre al que tenga mayor versión, por lo que al final del proceso no deben existir paquetes duplicados en la instantánea final.

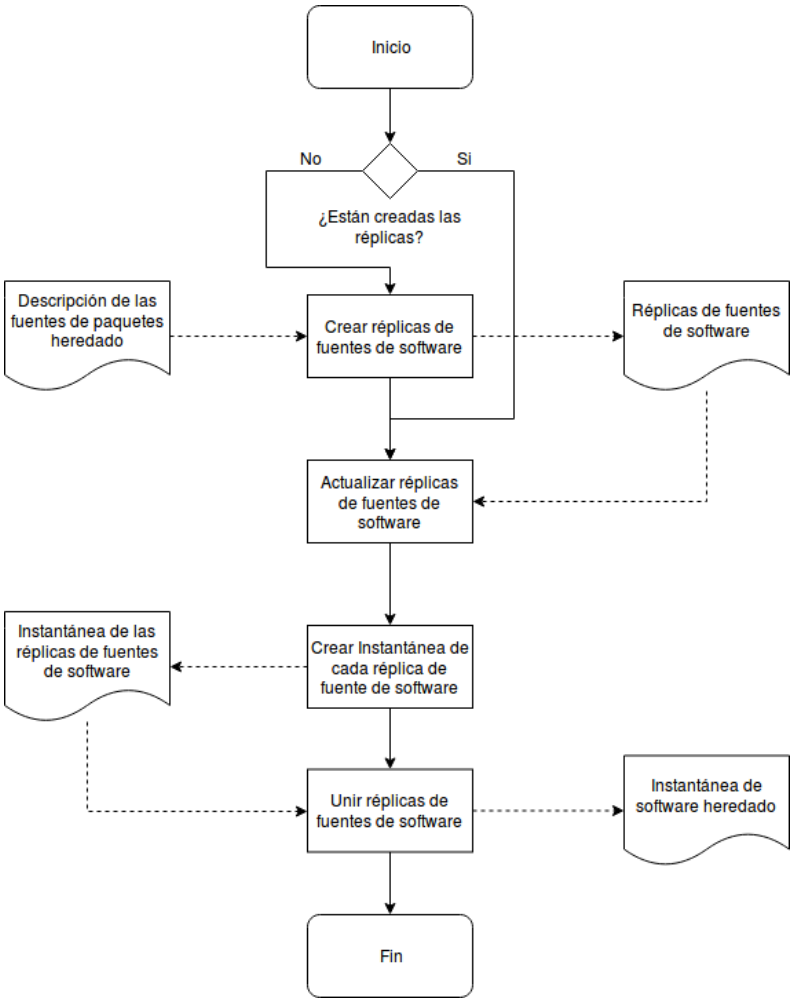


Fig. 11 Subproceso Crear instantánea de software heredado.

En la siguiente tabla se describen los elementos a tener en cuenta para completar el artefacto Descripción de fuentes de paquetes heredados.

Tabla 1 Elementos del artefacto Descripción de fuentes de paquetes heredados.

Campo	Descripción	Tipo
Nombre	Nombre de la réplica a crear.	Cadena de texto
URL	Dirección (web o física) del repositorio a hacer réplica.	Cadena de texto
Distribución	Nombre código de la distribución del repositorio a replicar.	Cadena de texto

Filtro	Nombre de los paquetes a replicar del repositorio en caso de que no se necesiten todos.	Lista de cadenas de texto
Arquitecturas	Arquitecturas a tener en cuenta para realizar la descarga	Lista de cadenas de texto

2.4.2 **Crear Instantánea de software propio**

Este paso tiene como entrada la Descripción de software propio (ver Tabla 2), en la cual se describe el repositorio de software que contiene todos los paquetes creados y modificados por Nova para una versión específica de la distribución. Con esta información se hace la lista de paquetes propios y se genera la Instantánea de software propio.

Tabla 2 Elementos del artefacto Descripción de software propio.

Campo	Descripción	Tipo
Nombre	Nombre del repositorio de software propio	Cadena de texto
Comentario	Descripción del repositorio de software propio	Cadena de texto
Componente	Componente del repositorio de software propio	Cadena de texto
Distribución	Distribución del repositorio de software propio	Cadena de texto
Arquitecturas	Arquitecturas de los paquetes presentes en el repositorio de software propio	Lista de cadenas de texto

2.4.3 **Subproceso Crear Instantánea de actualizaciones**

Una vez creadas las instantáneas de software propio y heredado es el momento de crear una instantánea con todo el software que debe ser actualizado. Debido a que generalmente el software heredado tiene versiones superiores a las del software propio es necesario realizar un reporte de actualización de paquetes que debe ser entregado a los desarrolladores para que actualicen los paquetes propios de la distribución y sean incluidos en el repositorio de software propio. Este reporte debe contener una tabla con los siguientes elementos:

- Nombre del paquete que se va a actualizar.
- Arquitectura del paquete.
- Versión en los repositorios de Nova.

- Versión en los repositorios de software heredado.

En este reporte deben ser incluidos también todos los paquetes que están en la instantánea de software heredado y no están los paquetes de Nova.

Teniendo en cuenta que los paquetes en un repositorio de distribuciones basadas en Debian siempre tienen un nombre con el formato <*nombre*>_<*versión*>_<*arquitectura*> con el siguiente ejemplo de una expresión regular de *sed* es posible detectar cuando un paquete común en las instantáneas de software heredado y propio es una actualización:

```
echo "apt_1.2.14_amd64" "apt_1.2.15_amd64" | sed -n 's/^\(.*\)_\(.*\)_\(.*\)\
\(.*\)_\(.*\)_\(.*\)/dpkg --compare-versions \2 lt \5 \&\& echo "\1 | \3 | \2 | \5"/ep'
```

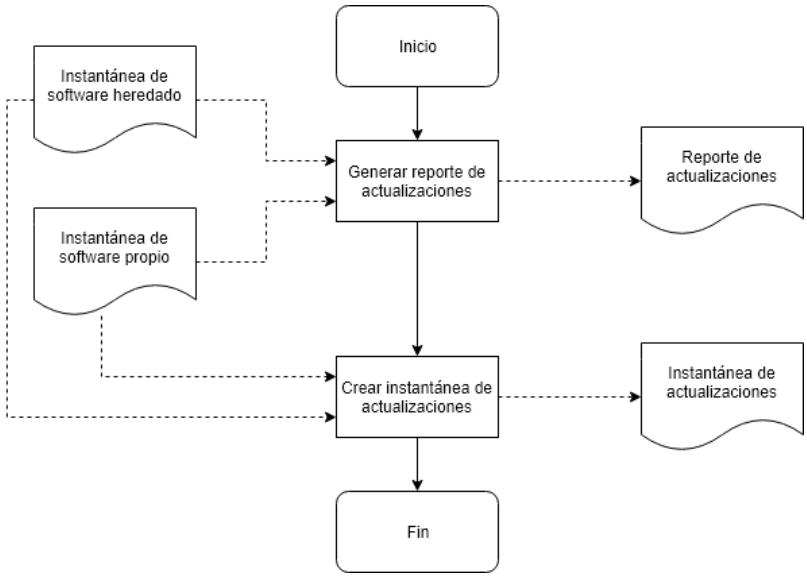


Fig. 12 Subproceso Crear Instantánea del repositorio de Nova.

Luego de creado el reporte se puede reutilizar esta información para generar una instantánea con solamente el código fuente de los paquetes que van a ser actualizados, para que los desarrolladores tengan un lugar para descargarlos, modificarlos (solo si sobrescribe algún paquete modificado previamente por Nova) y construirlos.

2.4.4 **Filtrar software con soporte oficial.**

Una vez constituida una instantánea con todo el software propio del repositorio de Nova, es necesario la creación de los componentes principal y extendido del repositorio. Este paso tiene como entrada la lista de paquetes con soporte oficial por parte del equipo de desarrollo.

Para la creación del componente principal se crea una instantánea que almacene todos los paquetes descritos en la lista de paquetes con soporte oficial y todas sus dependencias (recursivamente), propiciando que cada paquete incluido pueda ser instalado correctamente haciendo uso solamente de este componente. Generalmente esta lista de paquetes con soporte oficial es constituida por los paquetes binarios (nova-escritorio, nova-servidor y nova-ligero) generados a partir de la construcción del metapaquete de la distribución (nova-meta) y los paquetes del sistema operativo Linux. El componente extendido se construye a partir de la lista de paquetes resultantes de la diferencia entre la instantánea del repositorio completo y la del componente principal.

2.4.5 **Publicar ramas del repositorio.**

En el proceso se genera la estructura del repositorio lista para ser usadas por los gestores de paquetes. Para esto recibe como entrada las instantáneas de los componentes principal, extendido y actualizaciones, y la ruta donde va a ser creada la estructura en el sistema de archivos.

Otra de las entradas de este paso es el depósito de claves criptográficas utilizado para firmar los índices del repositorio y la información necesaria para completar el archivo *Release* listada a continuación:

Campo	Descripción	Tipo
label	Nombre de la distribución a la que pertenece el repositorio	Cadena de caracteres
origin	Origen del repositorio (puede ser el nombre de nombre de la distribución o la URL del mismo).	Cadena de caracteres
suite	Nombre de la distribución del repositorio (por ejemplo xenial-updates para Ubuntu o 2017 para Nova)	Cadena de caracteres (una palabra)
codename	Nombre de código de la versión de la distribución del repositorio.	Cadena de caracteres (una palabra)
version	Versión del repositorio	Cadena de caracteres (enteros separados por punto)
description	Breve descripción del repositorio	Cadena de caracteres

2.4.6 **Realizar verificación de integridad del repositorio.**

Mediante la verificación de integridad de los repositorios se pueden detectar errores ocurridos cuando se publicaron los repositorios y asegura que cada paquete contenido en los índices esté presente en la ruta correcta y con la suma de verificación adecuada. Para lograr esto es necesario verificar la integridad de los índices (comprobando que estén firmados adecuadamente) y los paquetes en el repositorio.

Basándose en las directivas de seguridad establecidas por *apt-secure* se puede realizar este procedimiento mediante una implementación del siguiente algoritmo:

```
Distribution = args[1]
BasePath = args[2]
DistsPath = BasePath + "/dists/" + Distribution
ReleaseFile = DistsPath + "/Release"
InReleaseFile = DistsPath + "/InRelease"
ReleasGPGFile = DistsPath + "/Release.gpg"

function SignatureVerification()
  if system("gpg --verify " + InReleaseFile)
    print("Signature verification ok")
  else
    print("Signature verification wrong")
    exit(1)

function VerifyMD5File(file, md5)
  if system("md5sum " + file + " | cut -d ' ' -f1") == md5
    print(file + " Hash verification ok")
  else
    print(file + " Hash verification wrong")

function VerifyComponent(file)
  for i in system("awk '/Package:\ /,/\n/' " + file) do
    Pkg = system("echo " + i + " | sed -n 's/^Package:\ \(.*\)$/\1/p'" )
    PkgFile = system("echo " + i + " | sed -n 's/^Filename:\ \(.*\)$/\1/p'" )
    PkgMD5 = system("echo " + i + " | sed -n 's/^MD5sum:\ \(.*\)$/\1/p'" )
    PkgFullPath = BasePath + "/" + PkgFile
    VerifyMD5File(PkgFullPath, PkgMD5)
  done

function DistsMD5Verification()
  DistsFiles = re("^\ [a-f0-9]{32}\ [0-9]+\ .+$",
                  read(ReleaseFile)).splitlines()
  for i in DistFiles do
    MD5 = i.split(" ")[1]
    File = DistsPath + "/" + i.split(" ") [3]
    VerifyMD5File(File, MD5)
    if re("^.+\/Packages$", File)
      VerifyComponent(File)
  done
```

2.4.7 **Realizar verificación de dependencias**

Este paso se centra en detectar aquellos paquetes que contienen dependencias incumplidas (de los campos *Depends* y *Recommends*) en los componentes principal y extendido publicados en pasos previos. Durante este proceso se genera el Reporte de dependencias fallidas, el cuál debe contener la siguiente información:

- Nombre del paquete con dependencia fallida.
- Lista de dependencias fallidas del paquete y sus causas.
- Arquitectura de hardware para la cual la dependencia no puede ser cumplida.

Además, son generados dos reportes sobre el componente de actualizaciones del repositorio publicado:

1. Reporte de dependencias de construcción correctas: contiene una lista de los paquetes de código fuente del componente de actualizaciones que tienen todas sus dependencias de construcción cumplidas y que están listos para ser construidos.
2. Reporte de dependencias de construcción fallidas: contiene los paquetes de código fuente del repositorio de actualizaciones que tienen dependencias de construcción incumplidas y que necesitan la inclusión o construcción de otros paquetes para poder ser construidos e incluidos en el repositorio. Este reporte tiene los mismos elementos y formato Reporte de dependencias fallidas.

2.5 Diseño de la aplicación de software para el procedimiento.

Para la materialización del procedimiento se desarrolló una aplicación que cumpliera cada uno de los pasos y subprocesos expuestos.

2.5.1 **Propuesta de solución.**

Se propone la creación de un software para la gestión de repositorios de Nova que interactúe directamente con la aplicación Aptly, ordenándole la descarga y actualización de las réplicas de software heredado, así como la gestión de los repositorios almacenados y su publicación. Es responsable de la generación de todas las instantáneas para asegurar la reversibilidad de las operaciones en caso de errores y genera todos los reportes de actualización y verificación para que los administradores puedan conocer el estado actual y la calidad del repositorio publicado.

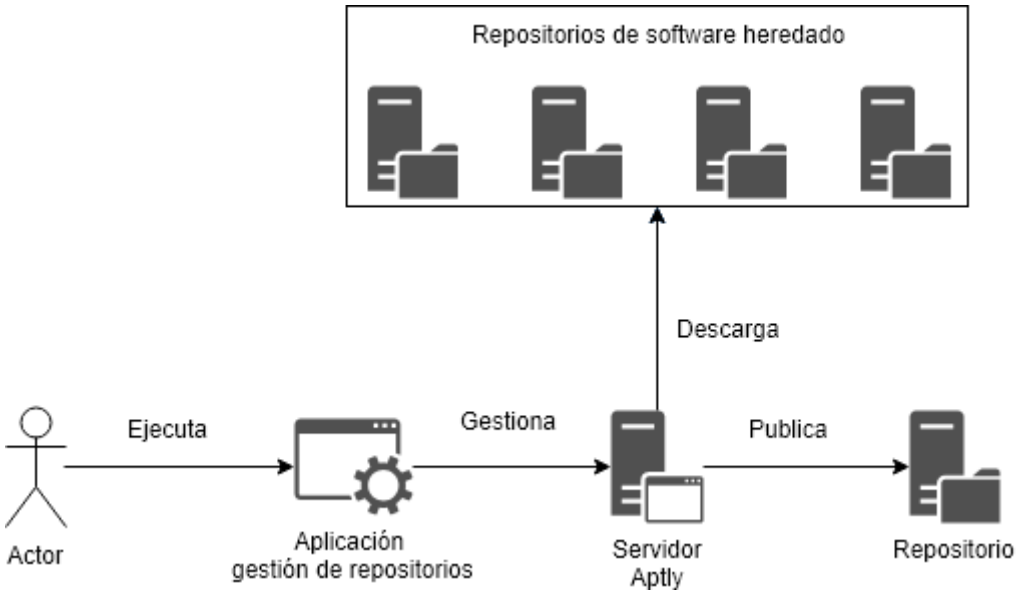


Fig. 13 Funcionamiento de la aplicación de gestión de repositorios de Nova.

2.5.2 **Arquitectura del sistema.**

Para el desarrollo de la arquitectura, se tuvo en cuenta el estilo arquitectónico Tuberías y filtros debido a que facilita en gran medida el desarrollo de la aplicación. En este estilo arquitectónico los componentes, llamados filtros, leen y procesan sus entradas para producir un flujo de salida. Existen diferentes tipos de filtros:

- Filtro fuente o *source filter*: producen un flujo de salida sin recibir ninguno de entrada
- Filtro transformador o *transform filter*: recibe un flujo de entrada, lo procesa y genera un flujo de salida.
- Filtro consumidor o *sink filter*: recibe un flujo de entrada y no produce un flujo de salida.

Los conectores del proceso son las tuberías que llevan la salida de un filtro hacia la entrada de otro. Este estilo arquitectónico tiene muchas ventajas como la facilidad ser descrito, entendido e implementado y su modularidad pues resultan irrelevantes los detalles de la implementación de cada uno de los filtros siempre que estén bien descritos sus entradas, salidas y objetivos (François 2003).

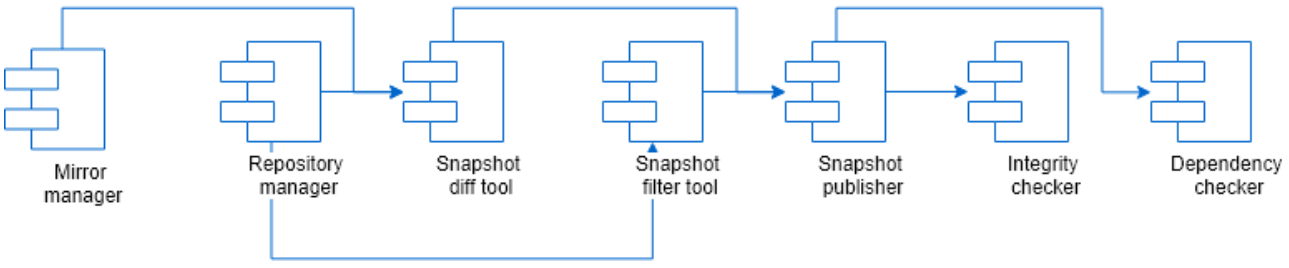


Fig. 14 Arquitectura de la aplicación de gestión de repositorios de Nova.

En la presente arquitectura todo el flujo de datos se realiza sobre las listas de paquetes, siendo los filtros los encargados procesarlos de acuerdo a lo establecido en el procedimiento expuesto anteriormente.

2.5.3 **Requisitos funcionales**

A continuación, se detallan los requisitos funcionales del sistema.

ID	Nombre	Descripción	Prioridad	Complejidad
RF1.	Gestionar instantáneas réplicas de software heredado.	Son creadas y actualizadas las réplicas de software heredado a partir de la descripción de las fuentes de software heredado. Se crean las instantáneas de cada una y se unen en una sola para crear la instantánea de software heredado.	Alta	Media
RF2.	Crear instantáneas repositorio de software propio.	Es creada la instantánea de software propio a partir de la descripción del repositorio de software propio.	Alta	Baja
RF3.	Generar instantánea de actualizaciones.	Es comparada la instantánea de software propio con la de heredado, generando la instantánea de actualizaciones con todos los paquetes fuentes nuevos y actualizables.	Alta	Alta
RF4.	Generar reporte de actualizaciones.	A partir de la instantánea de actualizaciones es generado un reporte de actualizaciones.	Alta	Baja
RF5.	Filtrar instantánea del repositorio de software propio.	Basado en la información de los paquetes primarios o metapaquetes de Nova y en la descripción de los componentes es filtrado la	Alta	Alta

		instantánea de software propio para crear una instantánea por cada componente de la distribución.		
RF6.	Publicar instantáneas del repositorio de software propio e instantáneas de actualizaciones.	Son publicadas las instantáneas de software propio y las instantáneas de actualizaciones, teniendo como entrada el depósito de claves de Nova, el identificador de la llave del archivo, la descripción de los componentes y la ruta donde se desea publicar el repositorio.	Alta	Media
RF7.	Realizar verificación de integridad.	Realiza verificaciones de integridad a los índices y paquetes teniendo como entrada la ruta donde están publicados, el depósito de claves de Nova y el identificador de la llave del archivo.	Alta	Alta
RF8.	Realizar verificación de dependencias de instalación.	Realiza verificación de las dependencias de instalación de los paquetes del repositorio teniendo como entrada la ruta donde están publicados.	Alta	Media
RF9.	Realizar verificación de dependencias de construcción.	Realiza verificación de las dependencias de construcción de los paquetes fuentes del repositorio teniendo como entrada la ruta donde están publicados.	Alta	Media

2.6 Conclusiones parciales

1. Se establecieron los criterios para de evaluación de las herramientas de gestión de repositorios mediante el Análisis Estructural, para luego evaluarlas utilizando la metodología QSOS, dando como resultado que Aptly tiene un mejor cubrimiento funcional que el resto.
2. Se definió un procedimiento para la gestión de los repositorios de Nova que contempla la verificación de integridad, verificación de dependencias y firma digital de los índices del mismo.
3. Se diseñó una aplicación, utilizando el estilo arquitectónico Tuberías y filtros, para que ejecute el procedimiento definido previamente.

Capítulo 3. Implementación y validación del modelo propuesto.

En el presente capítulo se exponen los detalles de la implementación de la solución propuesta, exponiendo el lenguaje de programación y el estándar de codificación utilizados, la integración con el servidor de automatización Jenkins y los detalles del despliegue. Por último, se analiza el estudio de caso de la gestión de repositorios de Nova para la arquitectura MIPS64EL de las computadoras Loongson utilizando el procedimiento propuesto y se analiza el índice de satisfacción grupal obtenido mediante la aplicación de la técnica Iadov.

3.1 Implementación.

A partir de los resultados obtenidos en la fase de análisis y diseño, se procede a materializar los mismos a código.

3.1.1 Lenguaje de programación.

Bash es un intérprete de líneas de comando (o Shell) para el sistema operativo GNU y es compatible con *sh* e incorpora características de Korn Shell (ksh) y C Shell (csh) y fue desarrollado para cumplir con el estándar IEEE POSIX P1003.2/ISO 9945.2 de Shell y Herramientas. Bash ofrece múltiples mejoras funcionales tanto para programación como de forma interactiva, entre las que se encuentran:

- Edición de la línea de comandos.
- Historial de comandos de tamaño ilimitado.
- Control de tareas.
- Funciones de Shell y uso de aliases.
- Arrays indexados de tamaño ilimitado.
- Aritmética de enteros en cualquier base desde dos hasta sesenta y cuatro.

Como casi todo el software de GNU, Bash es multiplataforma pudiéndose encontrar en casi todos los sistemas basados en Unix y MS-DOS.

3.1.2 Estándar de codificación.

Seguir un estándar de codificación consistente aumenta la calidad general de una aplicación de software y ayuda a la legibilidad del código escrito. Mientras más legible más fácil será para otra persona realizar acciones de mantenimiento o encauzar nuevos desarrollos y mejoras. Si a esto se

le suma que resulta mucho más sencillo encontrar y corregir errores en un código normalizado, entonces seguir un estándar de codificación no es algo a tomar a la ligera.

Para la implementación del sistema de gestión de repositorios se sigue el estándar de codificación establecido por el lenguaje de programación Bash para todos los proyectos que lo utilicen en Google (Armstrong 2014) y que se encuentra disponible en https://google.github.io/styleguide/shell.xml.

3.1.3 Integración con el servidor de automatización e integración continua Jenkins.

Jenkins Pipeline es un conjunto de complementos que permite implementar e integrar tuberías de entrega continua (o simplemente *pipeline*) en Jenkins. Un *pipeline* es una expresión automatizada de su proceso para obtener software desde el control de la versión hasta sus usuarios y clientes. Cada cambio en su software (cometido en control de versiones) pasa por un proceso complejo en camino a ser lanzado.

La definición de un *pipeline* de Jenkins está escrita en un archivo de texto (llamado *Jenkinsfile*) que a su vez puede ser asignado al repositorio de control de versiones de un proyecto o estar contenido dentro del mismo (*Pipeline-as-code*). A pesar de que los *pipelines* de Jenkins están diseñados para la entrega continua de aplicaciones desde el código fuente hasta el usuario, estos brindan todas las herramientas y potencialidades para facilitar la implementación de la solución propuesta. A continuación, se muestra la definición del *pipeline* para el procedimiento de gestión del repositorio de Nova.

```
node ("master") {
    stage("Update mirrors") {
        sh "repo-tool 2017 --update-mirrors"
    }
    stage("Snapshot Development Repositories") {
        sh "repo-tool 2017 --snapshot-repo"
    }
    stage("Find updates") {
        sh "repo-tool 2017 --find-updates"
        archiveArtifacts allowEmptyArchive: true, artifacts: "*.html,*.list", onlyIfSuccessful: true
    }
    stage("Filtring components") {
        sh "repo-tool 2017 --filter-components"
    }
    stage("Publishing repository") {
        sh "repo-tool 2017 --publish-components"
    }
    stage("Checking repository integrity") {
        sh "repo-tool 2017 --check"
        archiveArtifacts allowEmptyArchive: true, artifacts: "*.html, *.list, *.yaml", onlyIfSuccessful: true
    }
    stage("Checking repository dependencies") {
        sh "repo-tool 2017 --check-integrity"
        archiveArtifacts allowEmptyArchive: true, artifacts: "*.html, *.list, *.yaml", onlyIfSuccessful: true
    }
}
```

Fig. 15 Definición del pipeline con Jenkins

3.2 Despliegue de la solución propuesta.

3.2.1 Diagrama de despliegue

Para el correcto funcionamiento de la solución se propone el siguiente diagrama de despliegue. En él es necesario un servidor dedicado para la ejecución de Jenkins como servidor de automatización ejecutando la solución propuesta y que será brindado a través de servidor web Nginx. Además, se requiere un servidor para la ejecución de Aptly como herramienta de gestión de repositorios y con el cual el servidor con Jenkins se comunicará a través del protocolo SSH por el puerto 22. Por último, debe existir un servidor web de estáticos en el cual será utilizado para la publicación de la estructura del repositorio, siendo accedido a través del protocolo NFS por el puerto 111 por parte del servidor con Aptly.

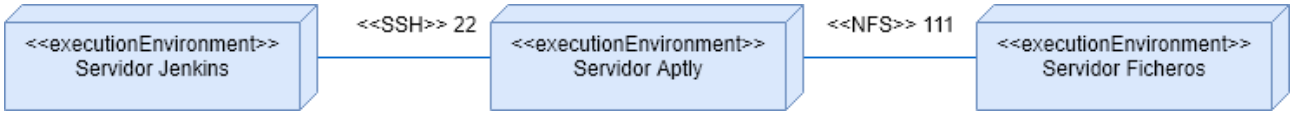


Fig. 16 Diagrama de despliegue de la solución.

3.2.2 Requerimientos de hardware.

A seguir, se desglosan los requerimientos de hardware recomendados por cada uno de los servidores descritos en el modelo de despliegue.

Tabla 3 Requerimientos de hardware para el despliegue de la solución.

Servidor	CPU	RAM	Disco duro
Jenkins	Dual Core 1.6 GHz o mayor	2 Gb	10 Gb
Aptly	Dual Core 1.6 GHz o mayor	2 Gb	400 Gb
Estáticos	Dual Core 1.6 GHz o mayor	2 Gb	400 Gb

3.2.3 Requerimientos de software.

Para el despliegue de la solución son necesarios los requerimientos de software siguientes:

- Sistema Operativo Nova Servidor 6.0
- Máquina virtual de Java 8 para el servidor Jenkins.

- Jenkins 2.121.1 con los complementos Pipeline y SSH Slave.
- Aptly 1.3.
- Servidor web Nginx 1.10.3.

3.3 Validación de la solución.

Para realizar la validación del procedimiento propuesto se aplica al proyecto Nova Loongson, incorporando la herramienta desarrollado al proceso de desarrollo y gestión del repositorio, y son analizados los resultados obtenidos y su influencia en el proyecto. Además, se aplica la técnica Iadov como método para conocer el estado de aceptación de los expertos y clientes con el procedimiento.

3.3.1 Validación funcional. Estudio de caso Nova Loongson.

El estudio de caso es una estrategia de investigación que ha sido asociada tradicionalmente con investigaciones cualitativas y de ciencias sociales, aunque existen investigaciones que demuestran su eficacia en la validación de métodos y herramientas informáticas (Kitchenham, Pickard, Pfleeger 1995) con el objetivo de asegurar que los cambios en los procesos provean los resultados deseados. Kitchenham et al. (1995) define los siguientes siete pasos a seguir para lograr diseñar y administrar un estudio de caso efectivo:

1. Plantear la hipótesis para definir el efecto a lograr.
2. Seleccionar los proyectos pilotos.
3. Identificar el método de comparación.
4. Minimizar el efecto de los factores difusos.
5. Planear el estudio de caso.
6. Monitorear el estudio de caso contra el plan.
7. Analizar y reportar los resultados.

3.3.1.1 Planteamiento de la hipótesis del estudio de caso.

El presente estudio de caso tuvo como objetivo demostrar la validez del procedimiento propuesto, su aplicabilidad en la gestión de los repositorios de Nova y la utilidad de los resultados obtenidos para aumentar los niveles de confiabilidad y seguridad de los repositorios de la distribución cubana de GNU/Linux. Cumpliendo con el primer paso para la elaboración del estudio de caso, la hipótesis definida fue que *el escalado del procedimiento de gestión de repositorios diseñado en el repositorio de Nova Loongson demostrará la validez del mismo para su aplicación en los repositorios oficiales*

de Nova, permitiendo elevar sus niveles de confiabilidad y seguridad. Durante la ejecución del estudio de caso se recogieron y analizaron los siguientes indicadores que fueron definidos en el procedimiento diseñado (ver Indicadores del estado del repositorio recolectados en el procedimiento.):

- Dependencias de dependencias de instalación fallidas.
- Dependencias de dependencias de construcción fallidas.
- Integridad de los índices.
- Integridad de los paquetes binarios.
- Integridad de los paquetes de código fuente.
- Autenticidad de los índices.

3.3.1.2 **Selección del proyecto piloto.**

El proyecto piloto seleccionado para el estudio de caso fue Nova Loongson en la fase de desarrollo, un proyecto de investigación-desarrollo que surge de la necesidad de proveer al país un sistema completo (hardware y software) de especificaciones libres en aras de aumentar la seguridad de las estaciones de trabajo de los Órganos y Organismos de la administración Central del Estado. El objetivo de Nova Loongson es proveer un sistema operativo para las computadoras con arquitectura de hardware MIPS64EL fabricadas por la empresa china Loongson, y en las que no pueden ser ejecutados los programas binarios convencionales (malignos o no) y solo pueden funcionar cuatro sistemas operativos: Linux Deepin, Fedora, Ubuntu y Nova, siendo todos variantes de GNU/Linux.

El repositorio del proyecto Nova Loongson contiene un conjunto de características que lo hacen un escenario de prueba ideal para esta investigación:

- Repositorio de software propio de mediano tamaño (14112 paquetes binarios y 4570 paquetes de código fuente para un total de 25 Gb) comparado con el de una versión de Nova para las computadoras Intel (83535 paquetes binarios y 26449 paquetes de código fuente para un total de 91 Gb), por lo que los tiempos de verificación de dependencias y de integridad y de publicación son mucho menores.
- Solamente dos repositorios de software heredado, el repositorio de Ubuntu desarrollado por los propios especialistas de Loongson y el repositorio de Gnome 3, este último solamente de código fuente. Esta situación reduce los tiempos de descarga y actualización de las réplicas de software heredado.

- Una sola arquitectura de hardware (MIPS64EL) por lo que solo es necesario realizar la verificación de dependencias de construcción e instalación una vez, a diferencia de Nova para x86_64 que tiene dos.
- Repositorio independiente del repositorio de Nova, lo que posibilita que cualquier acción tomada en este no tendrá ninguna repercusión en el repositorio oficial de la distribución.

Para la realización del estudio de caso fue escogido un repositorio más pequeño que el repositorio oficial de Nova (cerca de un cuarto de tamaño si se elimina una de las arquitecturas), con el objetivo de realizar el escalado del procedimiento. Una vez realizado el estudio de caso en los repositorios de Nova Loongson se induce que el comportamiento en los repositorios oficiales será proporcional debido a que la complejidad temporal de las operaciones del procedimiento es lineal y tiene como cota superior el número de paquetes en el repositorio y las fuentes de actualizaciones. Cabe destacar que el subproceso de “Crear instantánea de software heredado” depende directamente de circunstancias ajenas al procedimiento (velocidad de descarga de paquetes a través de Internet) por lo que no representa un medidor confiable para estimar la pertinencia del caso.

3.3.1.3 **Identificación del método de comparación.**

Para la realización del estudio de caso se creó un proyecto paralelo a Nova Loongson, reutilizando todos sus paquetes de software, al cual se le aplicó el procedimiento diseñado, para luego establecer una comparación entre el repositorio resultante y el original. Dentro de los métodos de comparación definidos por Kitchenham et al. (1995) fue seleccionado el de réplica del diseño del producto, siendo el equipo de desarrollo de Nova Loongson el responsable de tomar la medida de los indicadores seleccionados.

3.3.1.4 **Planeación el estudio de caso.**

Durante el análisis al proyecto original de Nova Loongson se detectó que ya se utilizaba Aptly como herramienta de gestión de repositorios, pero no existía un proceso definido y estandarizado para la gestión de los paquetes. Entre las principales deficiencias estaban:

- Todos los paquetes compilados por los desarrolladores eran subidos directamente al repositorio del proyecto y automáticamente publicados sin hacer una instantánea del estado anterior, lo que ocasionaba solapamiento de paquetes y pérdida de los mismos.
- Existían múltiples versiones publicadas de un mismo paquete.
- Mecanismo de entrada de actualizaciones sin definir, solo se había actualizado el repositorio de paquetes tres veces en un año.

- El repositorio estaba siendo firmado con una llave distinta a la oficial de Nova, por lo que los gestores de paquetes no lo reconocían como una fuente confiable.
- No se realizaban ningún tipo de verificaciones al repositorio, por lo que no existía un registro de que paquetes no podían ser instalados o construidos por problemas de dependencias y se desconocía si la generación del repositorio por parte de la herramienta era correcta.

3.3.1.5 **Análisis de los resultados.**

La aplicación del procedimiento fue realizada para los repositorios del repositorio del proyecto paralelo a Nova Loongson a partir de inicios de junio del 2018. Los resultados que se muestran a continuación corresponden a la primera ejecución de la herramienta.

Tabla 4 Resultados de la recolección de los indicadores en el estudio de caso Nova Loongson.

VERIFICACIÓN DE PAQUETES			
Indicador	**Fallidos**	**Total**	**% de fallo**
Dependencias de instalación fallidas	222	13289	1,67%
Dependencias de construcción fallidas	280	4459	6,28%
VERIFICACIÓN DE METADATOS			
Indicador	**Fallidos**	**Total**	**% de fallo**
Integridad de los índices	0	19	0%
Integridad de los paquetes binarios	0	13289	0%
Integridad de los paquetes fuentes	0	4459	0%
FIRMA CRIPTOGRÁFICA			
Indicador	**Fallidos**	**Total**	**% de fallo**
Índices firmados con la llave oficial	0	1	0%

Como se aprecia existen 222 paquetes presentes en el repositorio que no pueden ser instalados por los usuarios debido a errores de dependencias. A pesar de que el porciento error es bastante bajo, si se compara esta cantidad con el total de paquetes en el repositorio, no cuantifica el verdadero alcance del problema, pues si dentro de estos estuviese algún paquete declarado como esencial (paquete con prioridad *Required* o *Important*) o alguna dependencia de los metapaquetes (Escritorio, Ligero, Servidor o Base) de la distribución entonces ningún cliente podría tener instalado Nova en sus computadoras. Este número debe ser lo más cercano a cero posible.

En la verificación de las dependencias de construcción de los paquetes fuentes en el repositorio, se detectó la existencia de 280 paquetes que, a pesar de que fueron construidos previamente utilizando el mismo repositorio, no pueden volverse a construir debido a que ya no están presentes sus dependencias. Esta estadística no afecta al usuario del sistema operativo, pero si es muy importante para los desarrolladores porque afecta el indicador de Reproductibilidad (de Carné de Carnavalet,

Mannan 2014), el cual es muy importante para la seguridad de las distribuciones de GNU/Linux. Este número debe ser lo más cercano a cero posible.

La integridad de todos los paquetes y metadatos del repositorio fue verificada sin detectar alteraciones en la suma de verificación de los archivos, demostrando el buen funcionamiento de Aptly como herramienta de gestión de repositorios. Fue verificado además que el repositorio resultante en la aplicación del procedimiento fuera firmado con la llave oficial del archivo de Nova, paso que no se realizaba durante el desarrollo del proyecto Nova Loongson.

La implantación de este procedimiento en el proyecto Nova Loongson brindó información cuantitativa útil para mejorar la calidad del repositorio y beneficios en la mejora del proceso de desarrollo del producto como:

- Mecanismo de entrada de actualizaciones al repositorio: en el momento de ejecutado el procedimiento se descargaron 63 actualizaciones de paquetes que estaban pendientes desde la última vez que se actualizó el repositorio. Fue configurado la tarea en la herramienta Jenkins para que sea ejecutada periódicamente.
- Salvas automáticas del estado del repositorio: mediante la creación de instantáneas queda almacenado el estado del repositorio, brindando una vía para volver a estados previos ante errores en operaciones con el repositorio.
- Automatización y estandarización en el proceso: todos los pasos para la entrada de actualizaciones, creación de instantáneas, verificaciones, publicación y firma del repositorio se ejecutan de forma automática y estandarizada, y ya no dependen de la capacidad del personal para la realización de estas tareas.

Los resultados obtenidos en el estudio de caso realizado indican que la aplicación del procedimiento propuesto en Nova Loongson brindó información cuantitativa acerca de los indicadores seleccionados que puede ser utilizada para encauzar acciones para aumentar la confiabilidad y la seguridad de su repositorio, mejoró y estandarizó la gestión de los mismos y demostró su aplicabilidad en los repositorios de Nova, por lo que se acepta la hipótesis planteada.

3.3.2 **Evaluación de satisfacción grupal mediante el método Iadov.**

Con el objetivo de determinar el grado de satisfacción del cliente o público objetivo en cuanto a la utilidad del procedimiento y la herramienta en entornos reales de producción, se aplica la técnica Iadov (Kuzumina 1970). Este método permite la cuantificación de la satisfacción grupal de clientes o expertos con respecto a un proceso sujeto al análisis.

La técnica Iadov aplicada a esta investigación se basa en un cuestionario de cinco preguntas (tres cerradas y dos abiertas) a un grupo de expertos y clientes sobre el procedimiento y el software para

la gestión de repositorios de software. Una vez establecidas las preguntas se conforma el “cuadro lógico de Iadov” y el número resultante de la interrelación de las tres preguntas, indica la posición de los sujetos en la escala de satisfacción (Kuzumina 1970). La escala de satisfacción está dada por los criterios:

1. Clara satisfacción.
2. Más satisfecho que insatisfecho.
3. No definida.
4. Más insatisfecho que satisfecho.
5. Clara insatisfacción.
6. Contradictoria

Una vez obtenida escala de satisfacción de cada uno de los encuestados, se procede al cálculo de satisfacción grupal mediante la siguiente fórmula:

$$ISG = \frac{A(+1) + B(+0.5) + C(0) + D(-0.5) + E(-1)}{N}$$

Donde A, B, C, D, y E representan al número de encuestados con escala de satisfacción de 1, 2, 3 o 6, 4 y 5 respectivamente, y N representa al total de encuestados.

En la selección de los clientes se incluyeron los propios administradores de los repositorios de Nova, desarrolladores de la distribución y expertos en el área del desarrollo y migración a sistemas operativos y aplicaciones de software libre, y que cumplieran la condición de al menos cinco años de experiencia en la labor. Como resultado se conformó un grupo de diez especialistas, ocho pertenecientes a CESOL (centro de desarrollo de Nova) y dos pertenecientes al Centro de Soporte de la UCI.

A continuación, se muestra el cuadro lógico de Iadov generado a partir de la encuesta entregada a los expertos (ver Anexo 6: Encuesta para valorar la satisfacción de los clientes con el sistema de gestión de repositorios.).

Tabla 5 Cuadro lógico de Iadov.

<table>
<tr><td rowspan="2">5. Luego de ver el funcionamiento del sistema diga cuál es su nivel de</td><td colspan="3">3. Si pudieras escoger entre este procedimiento y el previo para la gestión de los repositorios ¿Utilizaría el previo para gestionar los repositorios de paquetes de software de su lugar de trabajo?</td></tr>
<tr><td>No</td><td>No se</td><td>Si</td></tr>
</table>

satisfacción respecto a él.	1. ¿Considera usted que el sistema cumple con las directivas de seguridad descritas en los estándares de los repositorios de Debian?								
	Si	No se	No	Si	No se	No	Si	No se	No
Me gusta mucho	1	2	6	2	2	6	6	6	6
No me gusta mucho	2	2	3	2	3	3	6	3	6
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	5
No sé qué decir	2	3	6	3	3	3	6	3	4

Luego de aplicadas la encuestas, se recogieron los niveles de satisfacción individual dando como resultado que el 80% de los encuestados estaban claramente satisfechos con el procedimiento y la aplicación desarrollados y el 20% restante estaban más satisfechos que insatisfecho, evidenciándose una clara aceptación individual con lo expuesto.

Nivel de satisfacción	**Cantidad**	**%**
Clara satisfacción.	8	80
Más satisfecho que insatisfecho.	2	20
No definida.	0	0
Más insatisfecho que satisfecho.	0	0
Clara insatisfacción.	0	0
Contradictoria	0	0

Para conocer el índice de satisfacción grupal se calcula mediante la fórmula antes descrita y puede arrojar resultados entre el -1 y el 1, mientras mayor sea el resultado mayor índice de satisfacción siendo 0,5 y 1 el rango de aceptación. Para esta investigación el índice de satisfacción grupal calculado es de **0,7**, por lo que se concluye que los clientes y expertos están satisfecho con la solución.

A pesar de que las preguntas abiertas realizadas no se tienen en cuenta para el cálculo del índice de satisfacción, si arrojan datos cualitativos que deben ser tenidos en cuenta para mejorar el procedimiento y encauzar futuras investigaciones y desarrollos. A continuación, se detallan algunas de los temas más destacados:

- Paralelización de las etapas del proceso.
- Mecanismo de construcción de paquetes para que, una vez terminado el proceso, compile todas las actualizaciones posibles y las adicione a un repositorio para ser probadas.
- Incorporar un algoritmo de ordenamiento que les dé prioridad a las actualizaciones que son dependencias de construcción de otras actualizaciones.

3.4 Conclusiones parciales.

1. Se materializó el procedimiento propuesto en una herramienta de gestión de repositorios, apoyándose en la herramienta de automatización Jenkins para facilitar la implementación de la arquitectura planteada y la ejecución de los diferentes pasos del procedimiento.
2. Se estableció el diagrama de despliegue, en el que se describen los servidores Jenkins, Aptly y Web de Ficheros Estáticos, y los protocolos de comunicación entre ellos. Además, se abordaron los recursos de hardware necesarios para la ejecución de la herramienta.
3. Se aplicó el procedimiento, mediante el uso de la herramienta desarrollada, al proceso de desarrollo del proyecto Nova Loongson, se analizaron las inconsistencias detectadas en el repositorio de software del proyecto y los beneficios aportados a su proceso de gestión.
4. Mediante la aplicación de la técnica Iadov se obtuvo un índice de satisfacción en un grupo de expertos seleccionados de 0.7 evidenciándose un alto nivel de aceptación del procedimiento expuesto.

Conclusiones

Mediante el estudio bibliográfico en el área de la gestión de los repositorios de Debian, se apreció la existencia de estándares para los mismos que rigen su estructura y definen sus principios de seguridad e integridad, así como múltiples procedimientos y herramientas para gestionarlos y verificarlos, las cuales fueron evaluadas utilizando la metodología QSOS dando como resultado que Aptly es la herramienta que brinda un mayor cubrimiento funcional.

Se diseñó un procedimiento para la gestión de los repositorios de Nova que permite la gestión de la entrada de actualizaciones y contempla la verificación de integridad y dependencias, y la firma digital de los índices, con el objetivo de aumentar la seguridad, integridad y confiabilidad del repositorio de Nova para los Órganos y Organismos de la Administración Central del Estado.

Mediante la implementación de una aplicación de software para la ejecución del procedimiento y el estudio de caso resultado de la aplicación de la misma sobre el proyecto Nova Loongson se comprobó su validez y su utilidad para encaminar acciones para aumentar la calidad del repositorio. Además, se aplicó la técnica Iadov y se obtuvo un índice de satisfacción en un grupo de expertos seleccionados de 0.7 evidenciándose un alto nivel de aceptación del procedimiento expuesto.

Recomendaciones

Para futuras investigaciones y desarrollos en la gestión de repositorios de Nova se realizan las siguientes recomendaciones:

- Definir métricas que permitan cuantificar la calidad de un repositorio basándose en el porciento de paquetes con dependencias fallidas (de construcción e instalación) y en su importancia para la distribución, y en la integridad del repositorio.
- Ejecutar de forma paralela las etapas del proceso que estén listas para ser ejecutadas para reducir el tiempo de ejecución.
- Incorporar un mecanismo de construcción de paquetes para que, una vez terminado el proceso, compile todas las actualizaciones posibles y las adicione a un repositorio para ser probadas.
- Incorporar un algoritmo de ordenamiento para la construcción que les dé prioridad a las actualizaciones que son dependencias de construcción de otras actualizaciones.

Bibliografía

ABATE, Pietro, 2014a. dose-distcheck(1) — dose-distcheck — Debian jessie — Debian Manpages. [online]. 2014. [Accessed 20 June 2018]. Available from: https://manpages.debian.org/jessie/dose-distcheck/dose-distcheck.1.en.html

ABATE, Pietro, 2014b. dose-builddebcheck(1) — dose-builddebcheck — Debian jessie — Debian Manpages. [online]. 2014. [Accessed 5 July 2018]. Available from: https://manpages.debian.org/jessie/dose-builddebcheck/dose-builddebcheck.1.en.html

ALBALAT, Miguel, FÍRVIDA, Abel, GARCÍA, Dairelys and MACHÍN, Jorge Luis, 2012. Nova ligero, incrementando la socio-adaptabilidad de la distribución cubana de gnu/linux. [online]. 2012. Available from: https://www.researchgate.net/profile/Dairelys_Garcia_Rivas/publication/283291919_NOVA_LIGERO_INCREMENTANDO_LA_SOCIO-ADAPTABILIDAD_DE_LA_DISTRIBUCION_CUBANA_DE_GNULINUX/links/5630eaf608ae0530378cf809/NOVA-LIGERO-INCREMENTANDO-LA-SOCIO-ADAPTABILIDAD-DE-LA-D

ARMSTRONG, Paul, 2014. Shell Style Guide. [online]. 2014. Available from: https://google.github.io/styleguide/shell.xml

BARTH, Andreas, DI CARLO, Adam, HERTZOG, Raphaël, NUSSBAUM, Lucas, SCHWARZ, Christian and JACKSON, Ian, 2017. Debian Developer's Reference. [online]. 2017. [Accessed 27 February 2018]. Available from: https://www.debian.org/doc/manuals/developers-reference/developers-reference.pdf

DEBIAN, 2013. Teams/FTPMaster - Debian Wiki. [online]. 2013. [Accessed 28 February 2018]. Available from: https://wiki.debian.org/Teams/FTPMaster

DEBIAN, 2017a. Distribución "testing" de Debian. [online]. 2017. [Accessed 14 March 2018]. Available from: https://www.debian.org/devel/testing

DEBIAN, 2017b. DebianDak - Debian Wiki. [online]. 2017. [Accessed 28 February 2018]. Available

from: https://wiki.debian.org/DebianDak

DEBIAN, 2018. DebianReleases - Debian Wiki. [online]. 2018. [Accessed 5 July 2018]. Available from: https://wiki.debian.org/DebianReleases#Workflow

DPKG DEVELOPERS, 2016. Ubuntu Manpage: dpkg-source - Debian source package (.dsc) manipulation tool. [online]. 2016. [Accessed 20 February 2018]. Available from: http://manpages.ubuntu.com/manpages/xenial/man1/dpkg-source.1.html

FERNÁNDEZ-SANGUINO, Javier, 2012. Securing Debian Manual - Debian Security Infrastructure. [online]. 2012. [Accessed 10 March 2018]. Available from: https://www.debian.org/doc/manuals/securing-debian-howto/ch7

FERNÁNDEZ-SANGUINO, Javier, 2016. The Debian GNU/Linux FAQ - Basics of the Debian package management system. [online]. 2016. [Accessed 20 February 2018]. Available from: https://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html#s-package

FORBES, Jonathan A., STONE, Jeremy D., PARTHASARATHY, Srivatsan, TOUTONGHI, Michael J. and SLIGER, Michael V., 1998. Software package management [online]. 19 June 1998. [Accessed 20 February 2018]. Available from: https://patents.google.com/patent/US6381742B2/en

FRANÇOIS, Alexandre R J, 2003. Software Architecture for Computer Vision: Beyond Pipes and Filters. [online]. 2003. [Accessed 7 July 2018]. Available from: http://www.dtic.mil/dtic/tr/fulltext/u2/a447811.pdf

GNUPG, 2018. The GNU Privacy Guard. [online]. 22 February 2018. [Accessed 9 March 2018]. Available from: https://www.gnupg.org/GnuPG is a free implementation of OpenPGP

GODET, Michel, MONTI, Régine, MEUNIER, Francis and ROUBELAT, Fabrice, 2000. LA CAJA DE HERRAMIENTAS DE LA PROSPECTIVA ESTRATÉGICA. [online]. 2000. [Accessed 8 June 2018]. Available from: http://www.asapbiblioteca.com.ar/wp-content/uploads/2014/10/cajadeherramientas_godet.pdf

HERTZOG, Raphaël and MAS, Roland, 2015. *The Debian Administrator's Handbook* [online]. [Accessed 20 February 2018]. ISBN 979-10-91414-05-0. Available from: http://www.gnu.org/licenses/.

JACKSON, Ian, 2017. Debian Policy Manual v4.1.3.0. [online]. 2017. [Accessed 21 February 2018]. Available from: https://www.debian.org/doc/debian-policy/

KARACHIWALA, Aslam, 2017. SecureApt - Debian Wiki. [online]. 2017. [Accessed 10 March 2018]. Available from: https://wiki.debian.org/SecureApt#Basic_concepts

KITCHENHAM, Barbara a., PICKARD, L. M. and PFLEEGER, Shari Lawrence, 1995. *Case studies for method and tool evaluation* [online]. 1995. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=391832

KUZUMINA, N.V., 1970. Metódicas investigativas de la actividad pedagógica. *Editorial Leningrado*. 1970.

LINK, Bernard R., 2013. Reprepro manual. [online]. 2013. [Accessed 2 March 2018]. Available from: http://mirror.physics.ox.ac.uk/mirror/ubuntu/oxford-local-testing/reprepro/docs/manual.html

LINK, Bernhard R., 2011. REPREPRO. [online]. 2011. [Accessed 20 February 2018]. Available from: https://mirrorer.alioth.debian.org/reprepro.1.html

LINK, Bernhard R., 2015. mirrorer/reprepro - reprepro - produce, manage and sync a local repository of Debian packages. [online]. 2015. [Accessed 20 February 2018]. Available from: https://anonscm.debian.org/cgit/mirrorer/reprepro.git/tree/docs/recovery

MANCINELLI, Fabio, BOENDER, Jaap, DI COSMO, Roberto, VOUILLON, Jérôme, DURAK, Berke, LEROY, Xavier and TREINEN, Ralf, 2006. Managing the complexity of large free and open source package-based software distributions. *Proceedings - 21st IEEE/ACM International Conference on Automated Software Engineering, ASE 2006*. 2006. No. May 2014, p. 199–208. DOI 10.1109/ASE.2006.49.

MANCOOSI, 2006. EDOS. [online]. 2006. [Accessed 5 July 2018]. Available from: http://www.mancoosi.org/edos/

PÉREZ HERRERA, Dariem, 2013. *Sistema distribuido para automatizar la construcción de repositorios de paquetes binarios de la distribución cubana de GNU/Linux Nova* [online]. Universidad Central “Marta Abreu” de Las Villas. [Accessed 20 February 2018]. Available from: http://dspace.uclv.edu.cu/bitstream/handle/123456789/7219/2013-01-31 Tesis_Nova_Distributed_Build_System.pdf?sequence=1&isAllowed=y

PIERRA FUENTES, Allan, 2011. *Nova, distribución cubana de GNU/Linux. Reestructuración estratégica de su proceso de desarrollo* [online]. Universidad de las Ciencias Informáticas. [Accessed 20 February 2018]. Available from: https://repositorio_institucional.uci.cu/jspui/bitstream/ident/8710/1/Allan Pierra Fuentes-TM.pdf

QSOS, 2017. Collaborative technological watch. [online]. 2017. [Accessed 10 March 2018]. Available from: http://www.qsos.org/

RABELO, Jesús, GONZÁLEZ, Marielis, PÉREZ, Héctor and DELGADO, Mairim, 2015. Sistema de Notificaciones para la Plataforma de Desarrollo de Integración Continua de la distribución cubana de GNU / Linux , Nova . Notification System for Development Platforms of Continuous Integration inside the Cuban Distribution Pattern of GNU / Lin. [online]. 2015. Available from: http://wsl.softwarelivre.org/2015/0014/notification-system-for-development-platforms-of-continuous-integration-inside-the-cuban-distribution-pattern-of-GNULinux-nova-wsl-2015.pdf

SMIRNOV, Andrey, 2013. aptly - Overview. [online]. 2013. [Accessed 9 March 2018]. Available from: https://www.aptly.info/doc/overview/

SMIRNOV, Andrey, 2016. aptly - Debian repository management tool. [online]. 2016. [Accessed 8 March 2018]. Available from: https://www.aptly.info/

TREINEN, Ralf, 2014. Debian QA: dose tools. [online]. 2014. [Accessed 5 July 2018]. Available from: https://qa.debian.org/dose/debcheck.html

UBUNTU, 2016a. UbuntuBackports - Community Help Wiki. [online]. 2016. [Accessed 1 April 2018]. Available from: https://help.ubuntu.com/community/UbuntuBackports

UBUNTU, 2016b. StableReleaseUpdates - Ubuntu Wiki. [online]. 2016. [Accessed 1 April 2018]. Available from: https://wiki.ubuntu.com/StableReleaseUpdates

UBUNTU, 2017. Repositories/Ubuntu - Community Help Wiki. [online]. 2017. [Accessed 1 April 2018]. Available from: https://help.ubuntu.com/community/Repositories/Ubuntu

UBUNTU SECURITY TEAM, 2017. SecurityTeam/FAQ - Ubuntu Wiki. [online]. 2017. [Accessed 1 April 2018]. Available from: https://wiki.ubuntu.com/SecurityTeam/FAQ

VASCO, Oficina Tecnica de apoyo al software Libre Gobierno, 2011. *Estudio Prospectiva Software Libre 2020 EUSKADI*. 2011. Oficina Técnica de apoyo al software libre en el Gobierno Vasco.

WAUGH, Jeff, 2008. Understanding the Ubuntu package repositories – Be the signal. [online]. 2008. [Accessed 1 April 2018]. Available from: https://bethesignal.org/blog/2008/03/31/understanding-the-ubuntu-package-repositories/

WEIMER, Florian, 2005. Migration to APT 0.6. [online]. 2005. [Accessed 10 March 2018]. Available from: http://www.enyo.de/fw/software/apt-secure/

YANG, Boyuan, 2017. DebianUnstable - Debian Wiki. [online]. 2017. [Accessed 13 March 2018]. Available from: https://wiki.debian.org/DebianUnstable#Does_sid_have_security_updates.3F

Anexos

Anexo 1: Lista de expertos y actores en la gestión de repositorios de software libre.

Nombre	Ocupación
Juan Manuel Fuentes Rodríguez	Jefe de Departamento de Sistema Operativo, Centro de Software Libre.
Allan Pierra Fuentes	Jefe de Centro de Soporte.
Yoandi Pérez Villazón	Jefe de Centro de Software Libre.
Luis Daniel Sierra Corredera	Jefe de proyecto y desarrollador principal de Nova para Loongson.
Haniel Cáceres Navarro	Arquitecto Principal de Nova
Yileni Hechavarría González	Jefe de Proyecto de Nova
Manuel Enrique Peiso Cruz	Administrador de la configuración y los repositorios de Nova
Neyvis González Trejo	Administrador de la configuración y los repositorios de Nova
Marlon Rodríguez García	Desarrollador principal de Nova Escritorio
Javier Piñeiro Cárdenas	Desarrollador principal de Nova Ligero
Aldy León García	Desarrollador de Nova
Nelio Veliz Pedraza	Jefe de Departamento Simays, Centro de Software Libre
Gustavo Quezada Arévalo	Desarrollador de Nova Servidor

Anexo 2: Lista de variables claves del análisis estructural.

N°	Long label	Short label	Description	Theme
1	Documentación	Doc	Documentación asociada a la herramienta.	
2	Facilidad de uso	Fac	Usabilidad de la herramienta y sencillez para usuarios inexpertos.	
3	Gestión de grandes repositorios	Big	Capacidad de la herramienta de gestionar múltiples repositorios con un gran número de paquetes.	
4	Integración con sistemas de firma digital.	PKI	Capacidad de la herramienta de utilizar sistemas de firmas digitales para firmar los índices.	
5	Recuperabilidad ante errores	Fix	Capacidad de la herramienta de recuperarse sin pérdida de datos ante errores.	
6	Réplicas de repositorios remotos	Mir	Capacidad de la herramienta de hacer réplica de repositorios remotos.	
7	Sistema de verificación de integridad y dependencias	Ver	Capacidad de la herramienta de brindar funcionalidades para verificar dependencias e integridad de los paquetes.	
8	Mecanismo de entrada de paquetes	Inc	Mecanismo de entrada de paquetes basándose en los changes generados por la compilación.	
9	Inclusión de paquetes sin código fuente	Cod	Posibildad de la herramienta de incluir paquetes sin el código fuente.	
10	Mecanismo de gestión de instantáneas	Sna	Capacidad de la herramienta de gestionar instantáneas del repositorio.	

N°	LONG LABEL	SHORT LABEL	DESCRIPTION	THEME
11	Integración con servicios externos de integración continua	CIS	Capacidad de la herramienta de brindar funcionalidades que le permitan integrarse con sistemas externos de integración continua.	
12	Posibilidad de inclusión de múltiples versiones de un mismo paquete	Mul	Capacidad de la herramienta de gestionar múltiples versiones de un paquete en el mismo repositorio.	

Anexo 3: Ponderación de los criterios de evaluación para la selección de la herramienta para la gestión de repositorios de paquetes de Debian.

Criterios de evaluación	Debian Archive Kit	Reprepro	Aptly
Documentación	0	1	2
Facilidad de uso	0	1	2
Gestión de grandes repositorios	2	1	1
Integración con sistemas de firma digital	2	1	2
Recuperabilidad ante errores	1	0	2
Capacidad de hacer réplicas de repositorios remotos	0	2	2
Sistema de verificación de	1	0	1

integridad y dependencias			
Mecanismo de entrada de paquetes	2	1	1
Mecanismo de gestión de instantáneas	0	1	2
Integración con servicios externos de integración continua	2	0	1
Inclusión de múltiples versiones de un mismo paquete	0	0	2
	10	8	18

Anexo 4: Cubrimiento funcional de los criterios de evaluación.

Criterios de evaluación	Cubrimiento funcional
Documentación	2
Facilidad de uso	2
Gestión de grandes repositorios	2
Integración con sistemas de firma digital	2
Recuperabilidad ante errores	2
Capacidad de hacer réplicas de repositorios remotos	2

Sistema de verificación de integridad y dependencias	1
Mecanismo de entrada de paquetes	2
Inclusión de paquetes sin código fuente	1
Mecanismo de gestión de instantáneas	2
Integración con servicios externos de integración continua	1
Inclusión de múltiples versiones de un mismo paquete	1

Anexo 5: Evaluación de las herramientas de gestión de repositorios aplicando QSOS.

Criterios de evaluación	Debian Archive Kit	Reprepro	Aptly
Documentación	0	2	4
Facilidad de uso	0	2	4
Gestión de grandes repositorios	4	2	2
Integración con sistemas de firma digital	4	2	4
Recuperabilidad ante errores	2	0	4

Capacidad de hacer réplicas de repositorios remotos	0	4	4
Sistema de verificación de integridad y dependencias	1	0	1
Mecanismo de entrada de paquetes	4	2	2
Mecanismo de gestión de instantáneas	0	2	4
Integración con servicios externos de integración continua	4	0	2
Inclusión de múltiples versiones de un mismo paquete	0	0	2

Anexo 6: Encuesta para valorar la satisfacción de los clientes con el sistema de gestión de repositorios.

La medida de la satisfacción del cliente con respecto al sistema propuesto, así como su utilidad y aplicabilidad en entornos reales, se llevará a cabo a partir del juicio que usted emita. Según su criterio, responda las siguientes preguntas:

1. ¿Considera usted que el sistema cumple con las directivas de seguridad descritas en los estándares de apt-secure? Marque con una X su selección

Si_____ No se _____ No_____

2. ¿Qué elementos usted añadiría al procedimiento para aumentar la calidad del repositorio de Nova?

_____
_____
_____
_____
_____
_____
_____

3. Si pudieras escoger entre este procedimiento y el previo para la gestión de los repositorios ¿Utilizaría este procedimiento para gestionar los repositorios de paquetes de software de su lugar de trabajo?

Si_____ No se _____ No_____

4. ¿Qué otras herramientas de gestión repositorios conoces?

_____
_____
_____
_____
_____
_____
_____

5. Luego de ver el funcionamiento del sistema, diga cuál es su nivel de satisfacción respecto a él.

____ Me gusta mucho.

____ No me gusta mucho.

____ Me da lo mismo.

____ Me disgusta más de lo que me gusta.

____ No me gusta nada.

____ No sé qué decir.