Universidad de las Ciencias Informáticas

Facultad 1

Compresión del repositorio de Nova

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Adrian Lozano Cruz

Tutora:

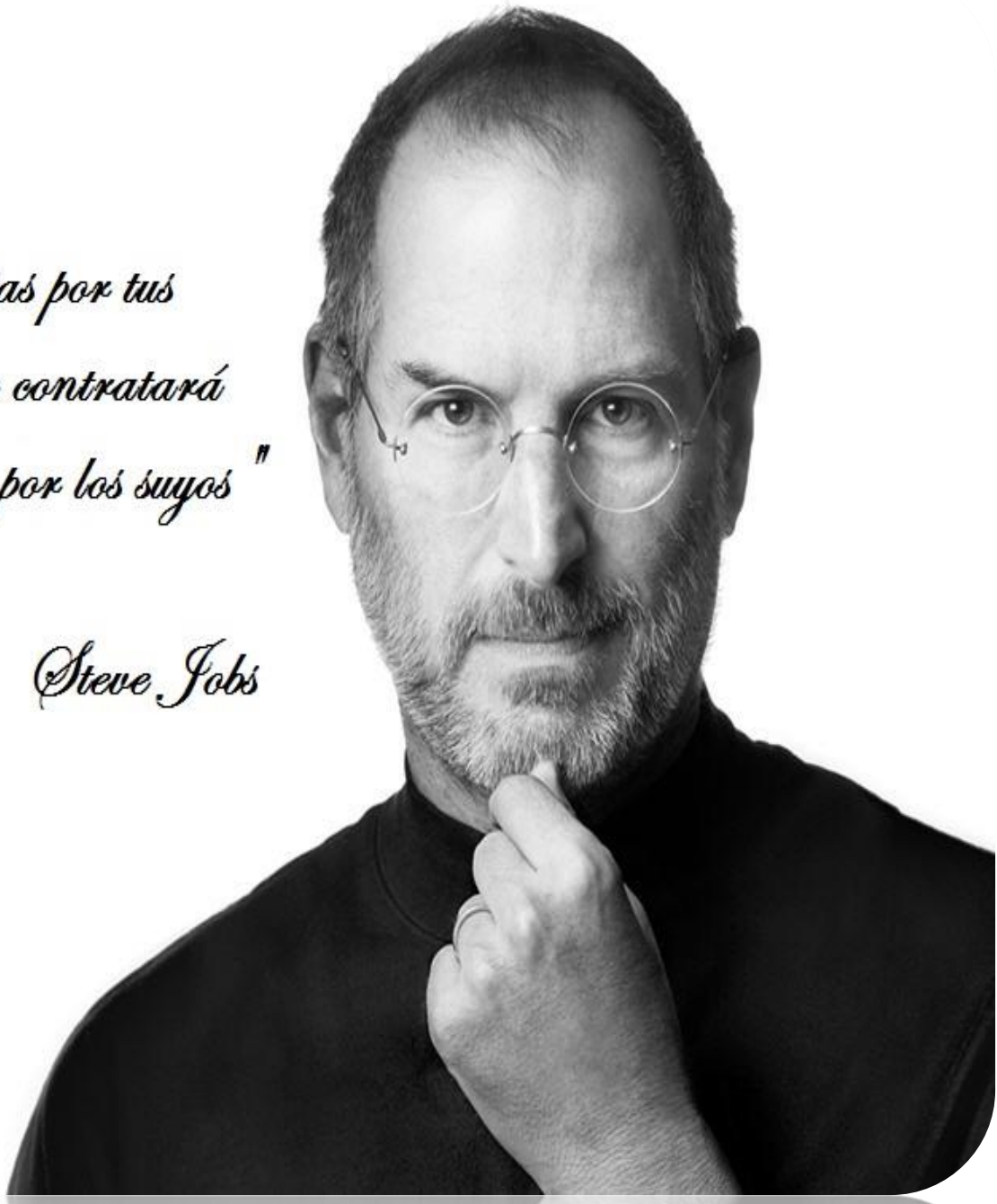
Ing. Dairelys García Rivas

La Habana, junio de 2016

"Año 58 de la Revolución"

"Si tú no trabajas por tus sueños, alguien te contratará para que trabajes por los suyos"

Steve Jobs



A mis padres y mi hermana, que siempre han estado cuando los he necesitado, gracias por darme otra oportunidad y confiar en mí. Los quiero mucho.

No alcanzaría la hoja para agradecerles a todas las personas que siempre se han preocupado por mí, pero quisiera agradecerle a mi novia por tener tanta paciencia conmigo y sus padres que siempre me han dado su apoyo. A mi tutora, porque sé que tener un tesista como yo no debe de ser fácil, gracias por estar siempre encima de mí cuando debería de haber sido al revés. A mis compañeros de aula que siempre me ayudaron cuando necesité de su ayuda, al equipo de la banda, los más malos del futbol sala en la UCI. A Yasmany por ser mi compañero de tesis, cuántas preguntas le habré hecho sin sentido, pero siempre estuvo ahí para ayudarme, al igual que Asiel (el lanza). A mi familia del casino que siempre se preocuparon por mis resultados, a todos los quiero mucho. Quiero agradecerle enormemente a mi familia de la Lenin, mis amigos incondicionales (Joan, Abdel, Vladimir, Boris, Yosvany, Mayito, Carlitos,

Aaron, y mi otra hermanita Ariadna), en las buenas y en las malas, tantos años juntos y seguiremos. Finalmente quiero agradecerles a mis sobrinas bellas, por demostrarme todos los días el amor y cariño que sienten hacia mí. A todos muchas gracias.

Declaración de autoría

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

_____

Autor: Adrian Lozano Cruz

_____

Tutora: Ing. Dairelys García Rivas

Datos del contacto

Tutora: Ing. Dairelys García Rivas.

Graduada de Ingeniería en Ciencias Informáticas en la UCI en el año 2012. Jefa del departamento de Sistema Operativo de Nova. Con 3 años de experiencia como ingeniera y 8 años de experiencia trabajando con *software* libre.

Correo electrónico: dgrivas@uci.cu

Actualmente las descargas del repositorio de la distribución cubana de GNU/Linux Nova son lentas, ya sea ocasionado por el tamaño que el mismo posee o debido al poco ancho de banda con el que cuentan las instituciones del país, provocando inconformidad por los clientes que la usan. Para darle respuesta a lo planteado anteriormente, la presente investigación consiste en desarrollar una aplicación que permita comprimir el repositorio de la distribución cubana de GNU/Linux Nova, de manera que disminuyan los tiempos de descarga del mismo. La investigación se organizó a partir de tres capítulos. Para el desarrollo de los mismos fueron utilizados métodos teóricos, empíricos y particulares; los cuales permitieron estudiar sistemas homólogos en función de agilizar el desarrollo del sistema. Como consecuencia se definieron las metodologías, herramientas y elementos de ingeniería de *software* más convenientes que permitieron darle solución al problema. Con el propósito de obtener una aplicación de calidad se realizaron pruebas a la aplicación que contribuyeron a su validación.

Palabras claves: repositorio, compresión, tiempo de descarga, socio-adaptabilidad.

Currently the download repository of Cuban distribution of GNU / Linux Nova are slow, either caused by the size that it owns or due to low bandwidth with which have the country's institutions, causing dissatisfaction by customers that used. To give response to the points made above, the present research is to develop an application that can compress the repository of Cuban distribution of GNU / Linux Nova, so as to reduce download times thereof. The research was organized from three chapters. For the development of these were used theoretical, empirical and particular methods; which they allowed studying homologous systems based on speed development of the system. As a result, the methodologies, tools and engineering elements most convenient software that allowed give solution to the problem defined. In order to obtain a quality application testing application that contributed to its validation they are performed.

Keywords: repository, compression, download time, social adaptability.

Índice

Índice de figuras

Índice de figuras

Introducción

Debido al bloqueo económico y financiero que mantiene el gobierno de Estados Unidos sobre Cuba, y al dominio que poseen las compañías privativas en el mercado internacional de sistemas operativos, al país se le hace imposible mantenerse a la altura del modelo consumista desarrollado por las economías capitalistas en el área de las Tecnologías de la Información y las Comunicaciones (TIC).

A raíz de tal situación, el programa rector de la informatización de la sociedad cubana plantea desde el año 2004 que la misma se hará utilizando solamente plataformas de *software* libre, incluyendo dentro del proceso a los Organismos de la Administración Central del Estado (OACE[1]).

En aras de apoyar dicho proyecto surge la distribución cubana GNU/Linux Nova, desarrollada en la Universidad de las Ciencias Informáticas (UCI), a través de un proyecto que lleva el mismo nombre. Nova se basa principalmente en las cuatro libertades del *software* libre y en las 4S del Msc. Allan Pierra Fuentes[2]: Seguridad, Soberanía Tecnológica, Socioadaptabilidad y Sostenibilidad; las cuales dictan las condiciones políticas, sociales y económicas que regirán el proceso de migración a *software* libre en Cuba y las cuales se definen de la siguiente manera (1):

- **Seguridad:** el modelo de desarrollo colaborativo que propone el movimiento de *software* libre, el acceso al código fuente y el exhaustivo proceso de revisión y auditoría de código garantizará un sistema seguro de ataques y sin puertas traseras.
- **Soberanía Tecnológica:** es la capacidad del país para desarrollarse en dicho campo en forma autónoma. No supone autarquía (independencia absoluta) sino capacidad de decidir sobre su uso y desarrollo.
- **Socio-adaptabilidad:** las bases tecnológicas para la informatización de Cuba, deben ser hechas por cubanos y para los cubanos, logrando inigualable adaptabilidad a las condiciones de nuestro País.

[1] Organismos de la Administración Central del Estado.

[2] Allan Pierra Fuentes: autor de los principios para el desarrollo, uso y aplicación de las TIC en el Gobierno.

- **Sostenibilidad**: la constante asimilación e investigación de las nuevas tecnologías, la planificación, los modelos novedosos de comercialización y el uso racional de los recursos humanos, materiales y naturales, garantizarán soluciones, vigencia y sostenibilidad a largo plazo.

Precisamente elevar la socio-adaptabilidad es uno de los objetivos en los que más se ha enfocado Nova, lo cual, teniendo en cuenta las carencias tecnológicas del país, le otorga un punto a favor en comparación con las otras distribuciones de *software* libre elaboradas en el mundo.

Como todas las distribuciones de GNU/Linux, se necesita de un repositorio para mantener actualizado el sistema con los últimos cambios realizados por los desarrolladores y corrección de errores en el código y *bugs* de seguridad. El repositorio de Nova actualmente consta de aproximadamente 80 Gb de tamaño, y de acuerdo a la capacidad de la infraestructura de banda ancha con la cuenta el país, provoca que el proceso de hacer "*mirror*[3]" del repositorio sea considerablemente lento.

Otro de los factores que retrasan el desarrollo informático es el límite del ancho de banda con el que cuenta el país, que gira entre los 64 Kbps y los 2 Mbps, provocando que las tareas enfocadas en la red sean morosas, especialmente las descargas del repositorio[4] de la distribución cubana GNU/Linux Nova. Aunque se han desarrollado acciones técnicas con respecto al mejoramiento del repositorio de la distribución cubana de GNU/Linux Nova para disminuir tal afectación, los clientes no quedan del todo satisfechos, haciendo que la demora en las descargas del repositorio atente contra la buena aceptación del sistema operativo cubano.

En el año 2012 (2) se planteó la posibilidad de confeccionar para Nova un repositorio que solamente contara con las actualizaciones realizadas a los paquetes, lo que disminuiría el tiempo de actualización al facilitar el tráfico en la red. Esto fue demostrado en dicha investigación, donde también se plantea que dicho procedimiento disminuiría el almacenamiento necesario en los repositorios espejos.

De acuerdo a lo planteado previamente, se ha identificado como **problema de investigación**: ¿Cómo reducir el tamaño del repositorio de la distribución cubana de GNU/Linux Nova para facilitar el acceso al mismo desde los repositorios espejos?

[3] Proceso mediante el cual se sincroniza el repositorio original hacia varios repositorios más cercanos a los lugares desde los que se accede, para facilitar el tráfico por la red.

[4] Conjunto de paquetes de software disponibles para una distribución GNU/Linux almacenados de forma centralizada.

Para dar respuesta al problema científico expuesto se asume como **objeto de estudio:** los mecanismos y herramientas que permitan la compresión y la agilización de descarga del repositorio de la distribución cubana de GNU/Linux Nova, definiendo el **campo de acción** en las herramientas que permitan la compresión y agilización de descarga del repositorio de Nova.

Para brindarle una solución efectiva al problema, se plantea como **objetivo general:** Desarrollar una aplicación que, mediante el uso de un algoritmo, permita la compresión de paquetes en el repositorio de la distribución cubana de GNU/Linux Nova.

A partir del objetivo general se determina los **objetivos específicos** siguientes:

1. Realizar un análisis documental sobre las distintas formas de compresión.
2. Definir los algoritmos, tecnologías, metodologías y herramientas para el desarrollo de la aplicación.
3. Desarrollar la aplicación de compresión del repositorio para la distribución cubana de GNU/Linux Nova.
4. Validar y demostrar estadísticamente los resultados obtenidos.

Tareas de investigación:

1. Estudio sobre los distintos formatos de compresión.
2. Estudio de las metodologías de desarrollo.
3. Selección del algoritmo de compresión, tecnologías y herramientas para el desarrollo de la aplicación.
4. Implementación de la aplicación de compresión del repositorio de Nova.
5. Validación de los resultados obtenidos.

La investigación se rige por la siguiente **idea a defender:** El desarrollo de una aplicación de compresión del repositorio de Nova, permitirá la disminución del tiempo de descarga de la misma, y así facilitar el aumento de la socio-adaptabilidad del país.

Para darle cumplimiento a las tareas planteadas se empleó los siguientes métodos científicos de investigación:

Métodos teóricos

- **Método Analítico-Sintético:** permitió descomponer el problema en sus partes para poder realizar un mejor análisis, tomando información de distintas fuentes bibliográficas y la extracción de los elementos más importantes que se relacionan con las herramientas que permitan la comprensión y agilización de los tiempos de descarga del repositorio de la distribución cubana de GNU/Linux Nova.

Métodos empíricos

- **Método de Observación:** posibilitó observar el comportamiento de las herramientas y métodos existentes que permitan reducir el tamaño del repositorio y la agilización de su descarga.
- **Método Experimental:** permitió probar las herramientas y procesos observados, adaptando las condiciones para obtener conocimientos sobre los resultados en determinadas pruebas a las descargas del repositorio en la distribución cubana de GNU/Linux Nova.

Métodos particulares

- **La entrevista:** se realizó con el propósito de obtener información, puntos de vistas e ideas que contribuyan al desarrollo de la investigación y aporten conocimientos específicos del tema.

Estructura del documento

En este apartado se describe brevemente el contenido de cada uno de los capítulos en los que se secciona el presente trabajo investigativo:

Capítulo 1: "Fundamentación teórica"; se puntualizan los principales conceptos relacionados con el tema, se caracterizan los principales algoritmos de compresión que existen, además de definir las tecnologías, metodologías y herramientas a utilizar en el desarrollo de la aplicación.

Capítulo 2: "Análisis y diseño de la aplicación"; en este capítulo se realiza una descripción general de la solución propuesta y su funcionamiento, y partiendo de ella se especifican los requisitos funcionales y no funcionales de la solución, los elementos fundamentales del diseño y de la arquitectura a tener en cuenta para el desarrollo de la aplicación.

Capítulo 3: "Implementación y pruebas"; se exponen los resultados obtenidos acerca de las pruebas realizadas, verificando así el correcto funcionamiento y utilidad de la misma para beneficio del sistema operativo cubano.

Capítulo 1: “Fundamentación teórica”.

Introducción

Con el objetivo de facilitar la comprensión del alcance de la investigación, en el presente capítulo se exponen conceptos fundamentales asociados al dominio del problema. Además se realiza un análisis de los diferentes algoritmos de compresión. Se realiza un estudio del arte de sistemas que emplean repositorios. Por último, se analizan las metodologías, tecnologías y herramientas utilizadas durante el ciclo de desarrollo de la solución que se propone.

1.1 Conceptos asociados

1.1.1 Repositorio

En los sistemas operativos GNU/Linux, un repositorio es una colección de paquetes de programas de una distribución de Linux específica que generalmente contiene archivos binarios precompilados los cuales pueden ser descargados e instalados por los usuarios de la distribución correspondiente. En la actualidad existen diferentes conceptos relacionados con el término, uno de los desarrolladores de Debian, Aaron Isotton, lo define de la siguiente manera: “Un repositorio es un grupo de paquetes organizados en un árbol de directorios especiales, los cuales contienen también ficheros adicionales que indican los índices y el chequeo de sumas de los paquetes” (3).

Otras distribuciones también hacen alusión práctica a lo que se llama repositorio de Linux. Por ejemplo Ubuntu, que lo define como: “servidores que almacenan un conjunto de paquetes de *software* y que son administrados por los gestores de programas” (4). También Fedora lo trata como: “una colección de *software* ordenado, clasificado y disponible para su uso con herramientas compatibles que lo usen para descargar y manipular *software”* (5).

De acuerdo con las referencias anteriores, se define para la investigación que los repositorios para distribuciones de GNU/Linux son: “un conjunto de paquetes de *software* disponibles para una determinada distribución almacenados de forma centralizada, estos se encuentran alojados de forma estructurada y organizados por índices de los paquetes que estén disponibles, asegurando su autenticidad y que no se

encuentren dañados, de manera tal que los usuarios puedan administrarlos usando gestores de paquetes que los transfieren por vía ftp[5] o http[6]".

1.1.2 Estructura del repositorio Nova

Los repositorios de Nova siguen la misma estructura de los de Ubuntu, los cuales están divididos en dos archivos fundamentales: *dist* y *pools*.

Los dos directorios cuentan con una estructura formada por los componentes: *main*, *restricted*, *multiverse* y *universe*, donde el directorio *dist* almacena en cada uno, los índices de paquetes que corresponden a las diversas versiones y arquitecturas que estén contenidas en el repositorio de una determinada distribución. *Pools* por su parte, archiva subsecciones ordenadas alfabéticamente, dentro de las cuales se encuentran los paquetes de las distribuciones. Cada componente almacena sus recursos en dependencia de la estabilidad, mantenedor y licencia de las liberaciones de las nuevas versiones (6).

Existen igualmente en la estructura del repositorio dos ficheros especiales llamados índices que son el *Packages.gz* para almacenar la información de los paquetes binarios y el *Sources.gz* para los de código fuente, de donde el gestor de paquetes encontrará la ruta de la aplicación solicitada por el usuario en el repositorio. Todos los datos para acceder al repositorio como su dirección[7], los componentes y tipos de paquetes a descargar, se indican en el archivo de configuración *sources.list*[8] de cada sistema.

1.1.3 Gestores de paquetes

Los gestores de paquetes son herramientas que permiten el proceso automatizado de instalación, actualización o eliminación de paquetes de *software* desde los repositorios, adicionando funciones especiales como son: comprobación de firma digital, gestionar dependencias según se necesiten y

[5] File Transfer Protocol (ftp), es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol, por sus siglas en inglés), basado en la arquitectura cliente servidor.

[6] Hypertext Transfer Protocol (http) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web (www).

[7] Dirección en la red del repositorio, se denota con la palabra url.

[8] Archivo de configuración del gestor de paquetes APT instalado por defecto en Ubuntu que se ubica en /etc/apt/sources.list

comprobar la suma de verificación del paquete. Entre los gestores de paquetes que acepta Nova se encuentran *dpkg*, *apt*, *aptitude* y *synaptic*, utilizando en dicha cadena, la aparición de características mejoradas del anterior gestor (7).

1.1.4 Parches de paquetes de *software*

Las aplicaciones deben ser capaces de cumplir ciertos requisitos que fallan en determinado momento debido a errores detectados y cambios en las funcionalidades o en las tecnologías utilizadas; por tal asunto es necesario desarrollar nuevas versiones que suplan las dificultades de la anterior.

Anteriormente tal situación solo podía ser resuelta sustituyendo el programa entero en un ordenador por una nueva versión, aunque sus diferencias fueran mínimas, fácilmente daban lugar a más cantidades de datos y la tediosa tarea de la reinstalación del *software*. Por tales motivos se crean los ficheros que almacenan diferencias entre versiones de una misma aplicación, para que solamente se instalaran en el ordenador los pequeños cambios que solucionen errores o añadan nuevas funcionalidades para un determinado programa, sin necesidad de reinstalarlo.

Por dicho razón surgen los parches o deltas, que según diccionarios para términos informáticos, se refiere a: “un trozo de código objeto que se inserta en un programa ejecutable como remedio temporal de un error” (8). Otras fuentes, como la investigación “Algoritmos para compresión delta y sincronización de archivos remotos”, coinciden en que es: “un esquema donde se describen las diferencias entre la vieja y la nueva versión de un archivo” (9). Por tal razón se define al parche o delta para la investigación, como: “un fichero que contiene el tamaño mínimo de información de la diferencia entre versiones de un mismo programa”. Puede ser aplicado a diferentes tipos de ficheros, como en el caso de Linux que lo usa para alojar los cambios que ocurren entre los paquetes de *software*.

Las herramientas para la creación de deltas tienen dos funcionalidades principales: una para crear parches y otra para aplicarlos. La creación de parches consiste en que se cree el archivo de diferencia entre el par de versiones de un mismo paquete. Por otra parte, la aplicación de un parche ya creado, se refiere a que con la misma herramienta, especificando determinadas opciones, se pueda crear la nueva versión del paquete .deb, idéntica a que si se tuviera la original, con solo el viejo paquete (ya sea instalado o no) y el parche.

1.2 Compresión de archivos

La comprensión de archivos se refiere a la reducción del tamaño de un fichero para evitar que este ocupe mucho espacio. De acuerdo con lo planteado, el autor de la investigación arriba a la conclusión de que la función principal es la reducción del tamaño empleándose la codificación, donde su principal objetivo es minimizar el volumen del fichero original.

1.2.1 Formatos de compresión

La distribución GNU/Linux dispone de varios formatos a la hora de comprimir archivos. Como ya se había expuesto, dichos formatos reducen el espacio en disco de los ficheros y hacen que su envío sea más cómodo por vía ftp o http.

Los más usados por la distribución GNU/Linux son tar.bz2, tar.gz y zip, aunque existen otros como tar.7z, 7z y rar (10).

Zip

El formato ZIP[9] fue creado originalmente por Phil Katz, fundador de PKWARE. Katz liberó al público la documentación técnica del formato ZIP, y lanzó al mismo tiempo la primera versión de PKZIP[10] en enero de 1989 (10).

ZIP es un formato de fichero bastante simple, que comprime cada uno de los archivos de forma separada. Comprimir cada archivo independientemente del resto de archivos comprimidos permite recuperar cada uno de los ficheros sin tener que leer el resto, lo que aumenta el rendimiento. El problema, es que el resultado de agrupar un número grande de pequeños archivos es siempre mayor que agrupar todos los archivos y comprimirlos como si fuera uno sólo.

ZIP soporta un sistema de cifrado simétrico basado en una clave única. Sin embargo, este sistema de cifrado es débil ante ataques de texto plano, ataque de diccionario y fuerza bruta (10).

[9] ZIP es un formato de compresión sin pérdida, muy utilizado para la compresión de datos como documentos, imágenes o programas.

[10] PKZIP es un software de compresión para las computadoras. Permite comprimir un fichero o varios ficheros en un fichero de archivo de formato Zip.

Tar.gz

El formato de archivo tar.gz fue creado originalmente por Mark Adler y Jean Loup Gailly a principios de la década de 1990 para su uso específico en el proyecto GNU. Su actualización más reciente tuvo lugar en junio de 2013. El formato de archivo y su aplicación de *software* asociada se desarrollaron para permitir la compresión y descompresión de archivos de tamaño mucho mayor al habitual. Aunque son similares a los archivos en formato ZIP, los archivos tar.gz pueden gestionar archivos muchos mayores. Otra diferencia es que los archivos ZIP pueden contener múltiples archivos en su compresión, mientras que la extensión tar.gz solo permite la compresión de un único archivo de gran tamaño (11).

Tar.bz2

Tar.bz2 es un programa libre desarrollado bajo la licencia BSD[11] que comprime y descomprime ficheros usando los algoritmos de compresión de *Burrows-Wheeler* y de codificación de *Huffman*. El porcentaje de compresión alcanzado depende del contenido del fichero a comprimir, pero por lo general es bastante mejor al de los compresores basados en el algoritmo LZ77/LZ78. Como contrapartida, bz2 emplea más memoria y más tiempo en su ejecución (11).

7z o 7zip

Es un programa creado por Igor Pavlov en el 2006 bajo la licencia GPL[12] utilizado para archivar, comprimir y descomprimir archivos con la extensión .7z o .7zip a un alto grado de compresión y sin pérdida de datos (12). Para instalarlo es necesario contar con el compresor y descompresor en modo consola p7zip disponible para las distribuciones GNU/Linux que en realidad es la versión en línea de comandos de 7zip (13).

Entre las características de 7z más importantes está que es compatible con formatos de compresión: zip, gzip, bzip2, tar, tiene arquitectura abierta, trabaja con enlaces simbólicos y tiene fuerte cifrado

[11] La licencia BSD es la licencia de software libre otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution,* por sus siglas en inglés). Toma su nombre del sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados por la Universidad de California en Berkeley.

[12] GPL (General Public License, por sus siglas en inglés), es una licencia creada por la Fundación de *Software* Libre, que está orientada principalmente a proteger la libre distribución, modificación y uso de *software*.

AES[13]-256. Funciona además con archivos de tamaños hasta 16000000000 GB y como ha quedado demostrado tiene alta relación de compresión en donde utiliza el algoritmo LZMA, versión mejorada y optimizada del algoritmo LZ77[14] (14).

1.2.2 Selección del formato de compresión

Para seleccionar el formato de compresión a utilizar en los paquetes que componen el repositorio de la distribución cubana GNU/Linux Nova se tuvo en cuenta la tesis de Sandra Arango Winograd: "Disminución del tiempo de descarga para el repositorio de la distribución GNU/Linux Nova", ya que el tamaño del repositorio es proporcional a la suma de cantidad de paquetes que contiene, pues los demás ficheros que almacena poseen un tamaño despreciable con respecto al total:

Tamaño de repositorio = Σ tamaño de los paquetes

La fórmula indica que si se reduce el tamaño de los paquetes que contiene el repositorio, se podrá reducir el tamaño total del repositorio, facilitando así su tráfico en la red.

A continuación, se realiza un estudio estadístico con los formatos de compresión más usados y soportados en la distribución cubana de GNU/Linux Nova, para determinar cuál de todos sería el adecuado para la compresión de paquetes. Para realizar dicha tarea se tomaron en cuenta las variables **tamaño de compresión**, **tiempo de compresión** y **tiempo de descompresión**, las cuales deben ser minimizadas y siendo la última variable fundamental para el usuario final.

1.2.2.1 Tamaño de compresión

La tabla 1 presenta la relación del tamaño comprimido (MB) de una muestra tomada del actual repositorio, así como los porcentajes obtenidos respectivamente. Dicho paquete posee un tamaño de 513.6 MB y contiene 47851 archivos.

[13] Advanced Encryption Standard (AES) es un tipo de encriptación para datos electrónicos.

[14] Algoritmo sin pérdida, se utiliza cuando la información a comprimir es crítica y no se puede perder información.

Tabla 1: Comparación de los tamaños de compresiones para cada formato.

	tar.gz	tar.bz2	tar.xz	.zip	7z	tar.7z
X	130	114.1	93.8	139.7	80.8	80.9
%	25.34	22.24	18.28	27.23	15.75	15.77

La siguiente gráfica muestra los resultados obtenidos en la tabla 1 para un mejor entendimiento.

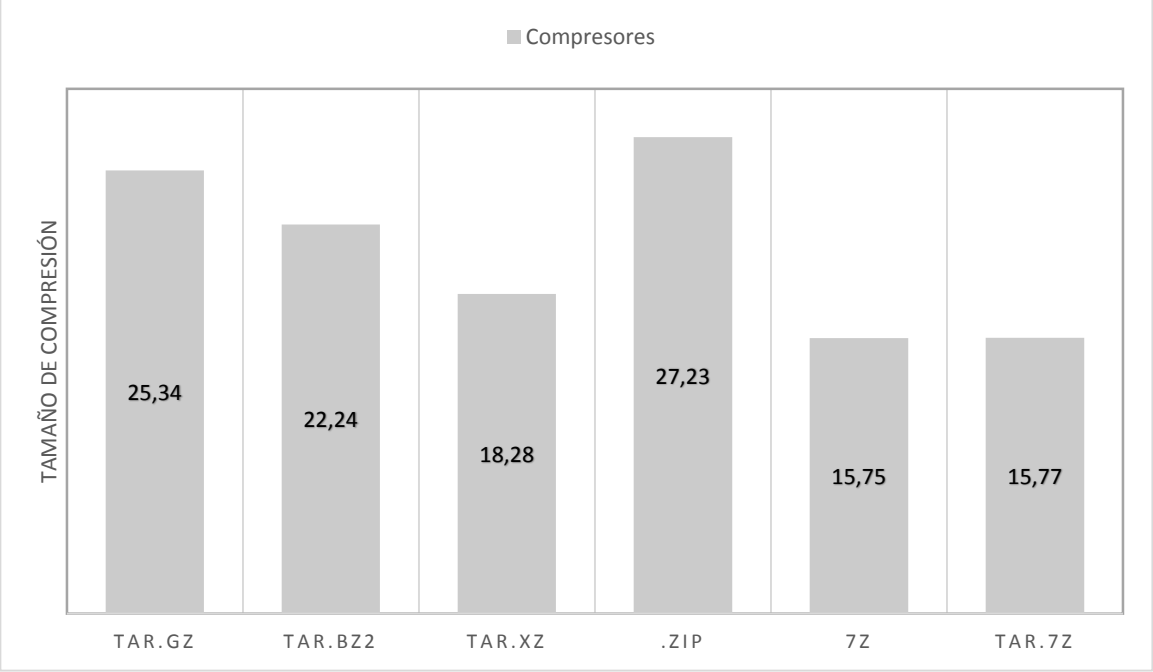


Figura 1: Porcentaje de compresiones para cada formato.

En la gráfica 1.1 se muestran los porcentajes de la media de los tamaños tras la compresión de la muestra para cada formato, ilustrando que se obtuvo una mejor compresión con 7z, tar.7z y tar.xz respectivamente.

1.2.2.2 Tiempo de compresión

A continuación, se presenta la comparación de los tiempos de compresión de los diferentes formatos, tomándose como media los tiempos en minutos que tarda en comprimir cada una de ellas con la misma muestra y los porcentajes que representa.

Tabla 2: Comparación de tiempos en minutos de compresión para cada formato.

	tar.gz	tar.bz2	tar.xz	.zip	7z	tar.7z
X	1:17	1:49	6:42	1:48	4:48	4:56
%	15.01	21.25	78.36	21.05	56.14	57.7

Según los resultados de los porcentajes de los tiempos en minutos que le tomó a cada herramienta realizar sus compresiones, se obtiene la gráfica siguiente:

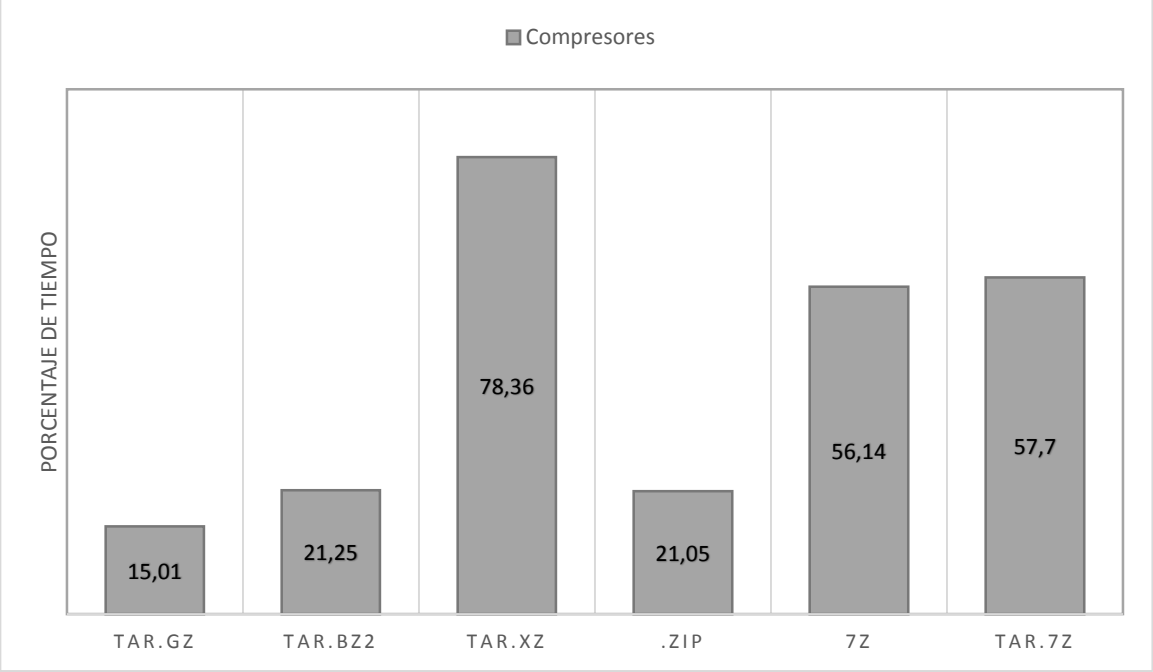


Figura 2: Porcentaje de tiempos totales de compresión para cada formato.

Según los resultados de los porcentajes en tiempo de compresión, los de menor tiempo en realizar la tarea son tar.gz y zip. Aunque de los tres algoritmos que mejor reducen el tamaño de la compresión, 7z es el que mantiene menor tiempo en la compresión.

1.2.2.3 Tiempo de descompresión

Se tomaron de igual forma la misma muestra y se realizan las comparaciones anteriores, mostrándose los porcentajes de las medias en minutos en tiempo de descompresión.

Tabla 3: Comparación de tiempos en minutos en descomprimir por cada formato.

	tar.gz	tar.bz2	tar.xz	.zip	7z	tar.7z
X	3:11	2:58	8:23	3:58	2:30	2:01
%	37.23	34.7	98.05	46.39	29.24	23.59

Resultados con más detalle en la siguiente gráfica.

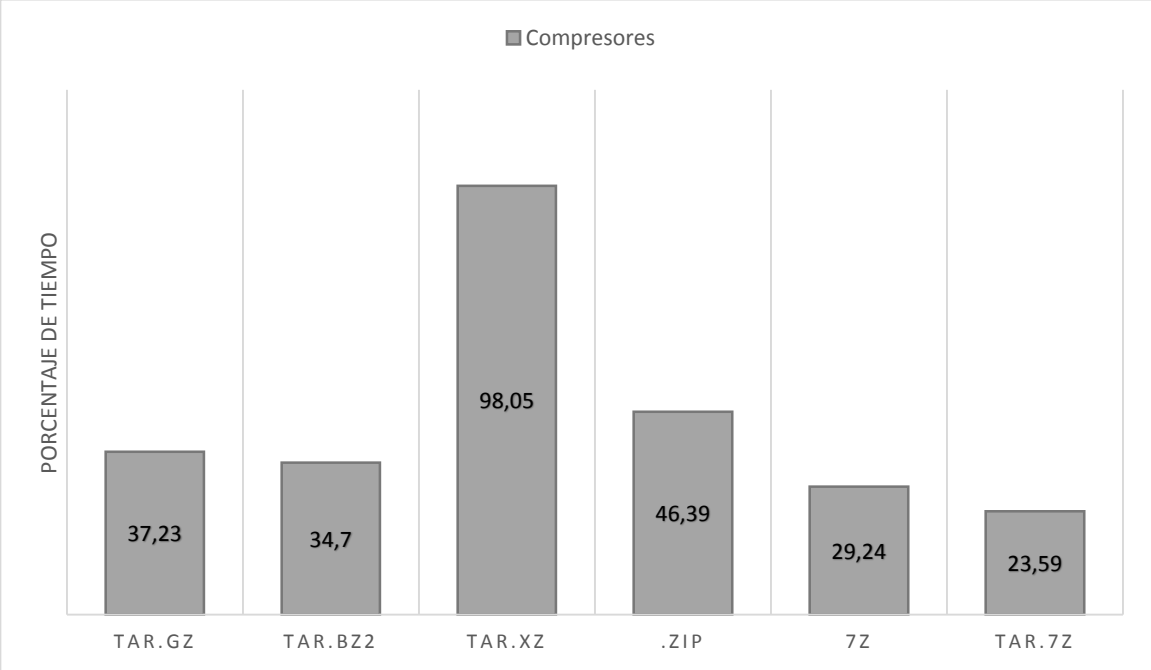


Figura 3: Porcentaje de tiempos totales de descompresión para cada formato.

Se observa que la mayoría de los algoritmos analizados tienen un alto grado de rapidez, exceptuando el caso de **tar.xz** en dicha operación. Pero de los algoritmos con mejores porcentajes en el tamaño de la compresión, **tar.7z** es el que menor tiempo ocupa en la descompresión.

1.2.2.4 Conclusiones del caso

Según el estudio realizado para la muestra aleatoria seleccionada y teniendo en cuenta las variables analizadas, el autor arriba a la siguiente conclusión:

Contemplando la necesidad de reducir el volumen del repositorio para la distribución cubana de GNU/Linux Nova, se forma una prioridad entre las variables: primero que el tamaño de compresión de los paquetes sea mínimo y luego que los tiempos requeridos para descompresión, que acelera el proceso de instalación para el usuario final, y la compresión sean igual de pequeños; todo en este orden.

En relación con lo antes planteado, los algoritmos con mejores resultados son 7z y tar.7z, al ser estos los de mejor tamaño en la compresión y descompresión de archivos.

De acuerdo al estudio de Arango Winograd, en una comparación más detallada entre 7z y tar.7z, el algoritmo 7z comprime un 10% más rápido que tar.7z, mientras que descomprimiendo tar.7z es un 45% más rápido que 7z (2).

Una ventaja distintiva de tar.7z es que emplea 7z archivándose con tar, haciendo que el contenido no pierda sus permisos y propiedades, situación que no maneja 7z y como además las diferencias entre los resultados de las herramientas no son las más alarmantes (2).

Por tal causa se llega a la conclusión de que la mejor opción a aplicar en la compresión paquetes del repositorio de la distribución cubana de GNU/Linux Nova es el algoritmo tar.7z tras la compilación de paquetes y poder tener los tamaños del paquete final lo más reducido posible.

1.3 Herramienta Debdelta para la creación de parches en el repositorio de Nova

Debdelta es un conjunto de aplicaciones para el trabajo con parches con extensión .debdelta que solo trabaja para paquetes binarios en sistemas Debian, creado por Andrea Mennucci bajo la licencia GNU[15] , liberado en mayo del 2006 (15).

La herramienta trabaja con un repositorio de actualizaciones y cuenta con programas fundamentales como: debdelta, que calcula la diferencia entre binarios y lo guarda en un archivo delta; debpatch que reconstruye una nueva versión de un paquete utilizando el parche y debdelta-upgrade que descarga deltas necesarias en los ordenadores de los usuarios finales.

[15] Licencia Pública General de GNU, es una licencia creada por la Free Software Foundation en 1989 y está orientada a proteger la libre distribución, modificación y uso de software.

Ventajas notables que añade debdelta es que cuenta con la restricción de que si el parche contiene una diferencia de un 70% de una versión con respecto a otra, entonces no se construye el delta, lo que supone una utilidad provechosa para el tamaño final de un repositorio de actualizaciones.

Cuenta con la opción distintiva de dar la posibilidad de usar conjuntamente con su compresión, algoritmos de distintas herramientas como xdelta, xdelta3 y bsdiff. Esta característica hace que la ejecución de debdelta se ralentice un poco más de lo normal. Pero para contrarrestar, cuenta con la alternativa de poder especificar el máximo de memoria que va a usar bsdiff o xdelta, aunque con el uso de ambas, se logra una mejor compresión en los parches resultantes (15).

Seguridad:

Entre sus funcionalidades cuenta al igual que xdelta, con la verificación de suma MD5[16] y se le añade GnuPG[17] y SHA1[18] a las opciones que se le pueden especificar (16).

Un paquete Debian que se crea con el delta descargado, es byte por byte idéntico al original, de modo que el soporte de autenticación criptográfica puede ser utilizado para afirmar que se puede confiar en que se instale un paquete original, a pesar de ser creado a partir de un parche.

Formas de uso:

Para explotar al máximo la herramienta debdelta es necesario tener un repositorio de parches que contendría los deltas de los paquetes del repositorio original y al cual accederían los usuarios para efectuar sus actualizaciones. Para cumplir la estrategia descrita, se debe contar con un servidor, que entre los recursos que puede almacenar, estén los repositorios de parches binarios. Cada parche es nombrado según una nomenclatura establecida por la distribución, especificando el nombre del paquete, el par de versiones para las que está hecha la diferencia, donde pueden existir tantos parches como pares de

[16] Algoritmo de reducción criptográfico.

[17] Herramienta que permite encriptar y firmar sus datos por contar con un versátil sistema de gestión de claves, lo que permite la seguridad en los parches creados.

[18] Sistema de funciones hash criptográficas, que se refiere a una función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc. Además de resumir o identificar un dato a través de la probabilidad, utilizando una función hash o algoritmo hash.

versiones del paquete que existan y por último la especificación de la arquitectura. Es por tal principio que la única diferencia de la estructura entre repositorios de paquetes de *software* y de parches, está en que estos últimos no necesitan índices, pues en el algoritmo de debdelta, la información que debería tomar de un índice la toma del mismo nombre del parche.

De forma paralela, el usuario final en su ordenador puede descargar desde el servidor del repositorio de parches, los deltas con la orden debdelta-upgrade, que es la encargada de descargar tanto los nuevos parches de los paquetes instalados que se registran en la caché, como descargar del repositorio habitual los paquetes binarios a actualizar para los que no se haya podido realizar un parche. Luego de la descarga, se aplican los deltas descargados a las viejas versiones instaladas y se crean entonces los nuevos paquetes .deb correspondientes.

De manera más explícita, cuando el usuario necesite actualizar un paquete, se recomienda utilizar inicialmente *apt-update* para que en la caché esté registrado la información de todos los paquetes disponibles para actualizar, lo cual serviría como guía a debdelta para buscar los parches necesarios. Después utilizamos *debdelta-upgrade* para descargar todos los deltas disponibles y los aplicaría construyendo los nuevos paquetes .deb que almacena en la cache de apt, en caso de ejecutarse como *root*, o en el archivo temporal tmp, si lo ejecutamos como un usuario sin privilegios. Una vez realizada esta descarga, con la llamada a *apt-upgrade*, ya se dispondrían de los nuevos paquetes para la actualización y los instalaría desde el mismo sistema sin necesidad de acceder al repositorio, siendo esta manera mucho más rápida que si no se hubiesen descargado antes los parches (15).

1.4 Análisis de tecnologías, metodologías y herramientas.

1.4.1 Metodología de desarrollo.

Una metodología de desarrollo de *software* representa un marco de trabajo que tiene entre sus funciones guiar, planificar, estructurar, controlar, manipular y dirigir el proceso de desarrollo de sistemas de información. Surge ante la necesidad de trabajar mediante el uso de procedimientos, técnicas, herramientas y documentos durante el desarrollo de *software*. Las metodologías se clasifican en dos tipos: ágiles o ligeras y pesadas o tradicionales. Las ágiles tienen como principios el trabajo en equipo como arma fundamental; el avance del trabajo enmarcándose solamente en los elementos necesarios que este exige. Se enfocan en la constante interacción con el cliente haciéndolo parte del equipo de trabajo y en la

posibilidad de cambiar todo lo que debe ser cambiado de forma que se alcance la mayor fiabilidad y calidad en el producto que se desarrolla. Por su parte, las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del *software*, con el fin de conseguir un *software* más eficiente. Estas se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada (17).

Programación Extrema

Programación Extrema (XP por sus siglas en inglés) es una metodología de desarrollo de *software* ágil que potencia las relaciones interpersonales del equipo de trabajo, confiándole a esta unión la clave del éxito en el desarrollo. El cliente juega un papel fundamental y decisivo durante el proceso, la retroalimentación entre él y el equipo de trabajo permite determinar de qué forma se va a implementar el trabajo durante todo el proceso de construcción del *software*. XP se define principalmente para proyectos cambiantes donde el trabajo constante traerá consigo cambios en el desarrollo de la solución para lograr siempre una mejor alternativa (18) (19).

Proceso Unificado de Desarrollo (RUP por sus siglas en inglés)

Por su parte, las metodologías pesadas se centran en el detalle de cada proceso y tarea que se debe desarrollar, en las herramientas a utilizar, genera una documentación extensa que debe justificar a cada paso de avance y además adelanta la puesta en práctica de la solución. Se aplica fundamentalmente a proyectos grandes para realizar en igual período de tiempo y uso de recursos. RUP define fases, principios, etapas o flujos de trabajo, disciplinas de soporte, entre otros elementos, para mejorar y organizar el trabajo desde el comienzo (20).

OpenUp

OpenUp es un proceso ágil y unificado, que contiene el conjunto mínimo de prácticas que ayudan a los equipos a ser más eficaces en el desarrollo de *software*. OpenUp es ágil, pues se centra en la naturaleza colaborativa de desarrollo de *software*. Es un proceso iterativo que es Mínimo, Completo y Extensible que puede utilizarse tal cual o ampliarse para tratar una amplia variedad de tipos de proyecto. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Está organizada dentro de cuatro áreas principales de contenido: Comunicación y Colaboración, Intención,

Solución y por último Administración. El OpenUp está organizado en dos dimensiones diferentes pero interrelacionadas: el método y el proceso.

El contenido del método es donde los elementos del método (roles, tareas, artefactos y lineamientos) son definidos, sin tener en cuenta como son utilizados en el ciclo de vida del proyecto. El proceso es donde los elementos del método son aplicados de forma ordenada en el tiempo. Muchos ciclos de vida para diferentes proyectos pueden ser creados a partir del mismo conjunto de elementos del método (21).

Beneficios en el uso del OpenUp:

1. Es apropiado para proyectos pequeños y de bajos recursos ya que permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.
2. Permite detectar errores tempranos a través de un ciclo iterativo.
3. Evita la elaboración de documentación, diagramas e iteraciones innecesarias requeridas en la metodología RUP.
4. Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

Fases que propone la metodología OpenUp:

1. **Concepción:** el objetivo de ésta fase es capturar las necesidades de los *stakeholders*[19] en los objetivos del ciclo de vida para el proyecto.
2. **Elaboración:** el propósito de esta fase es establecer la base la elaboración de la arquitectura del sistema y proporcionar una base estable para el gran esfuerzo de desarrollo de la siguiente fase.
3. **Construcción:** esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.
4. **Transición:** el propósito de esta fase es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y performance del último entregable de la fase de construcción.

[19] Se refiere a las personas involucradas en el desarrollo del *software*.

Finalmente, OpenUp es un proceso modelo y extensible, dirigido a la gestión y desarrollo de proyectos de *software* basados en desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo (22).

1.4.2 Selección de la metodología

Una vez analizadas estas metodologías se concluye que para guiar el proceso de desarrollo en cuestión se debe usar OpenUp. Se tuvo en cuenta para esta selección los beneficios expuestos anteriormente, además de ser apropiada para proyectos pequeños. También por ser una metodología que permite detectar errores tempranos a través de un ciclo iterativo.

1.4.3 Modelado

El lenguaje de modelado es vital para el diseño y posterior construcción del *software*. Es un conjunto estandarizado de notaciones que incluye símbolos y las distintas formas de organizarlos, estructurarlos y disponerlos lógicamente. Los diseños de *software* obtenidos son orientados a objetos.
El Lenguaje de Modelado Unificado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema. Describe el funcionamiento del mismo, sin profundizar en su implementación (23).

Herramienta CASE para el modelado

Las herramientas CASE (*Computer Aided Software Engineering*, siglas en inglés) son programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software* (24).

Este tipo de herramientas facilita el proceso de desarrollo de *software* con el aumento de la productividad y la reducción del tiempo que se utiliza. Utilizándolas se puede diseñar, implementar a partir del diseño realizado, compilar automáticamente, detectar errores y brindarle seguridad al equipo de trabajo sobre el avance de la solución. Para la selección de la herramienta más apropiada se han determinado una serie de aspectos que debe cumplir: modelar el *software* a través de diagramas UML, generar código en el lenguaje que se decida utilizar y la integración con diferentes IDEs (*Integrated Development Environment*, por sus siglas en inglés).

Visual Paradigm para UML

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida del desarrollo del *software* y posibilita un ahorro considerable de tiempo y una calidad óptima en el proceso. Se considera muy completa y fácil de usar, con soporte multiplataforma y excelentes facilidades de interoperabilidad con otras aplicaciones. Posibilita la captura de requisitos, análisis, diseño e implementación para una aplicación en desarrollo. Se decide utilizar esta herramienta para el modelado por las ventajas que ofrece (25):

- Soporte para UML versión 2.1: con la selección de la metodología OpenUp esta característica es muy provechosa ya que se necesita modelar los diagramas con el uso de UML.
- Generación de código: modelo a código, diagrama a código, para diferentes lenguajes, entre ellos Python.
- Generación de bases de datos: permite la generación automática de bases de datos a partir de un modelo entidad-relación.
- Interoperabilidad entre diagramas: permite a partir de un diagrama obtener otro que guarde relación con el mismo.
- Tiene apoyo adicional en cuanto a generación de artefactos automáticamente.
- Disponibilidad en múltiples plataformas: Microsoft Windows (98, 2000, XP, o Vista), Linux, Mac OS X, Solaris o Java.
- Generación de documentación: brinda la posibilidad de documentar todo el trabajo sin necesidad de utilizar herramientas externas.

Lo anteriormente expuesto demuestra que Visual Paradigm resulta de gran ayuda para un desarrollo exitoso del sistema que se implementa, ya que agiliza el proceso de desarrollo y genera los estereotipos necesarios con la estructura y relaciones deseadas.

1.4.4 Lenguajes de programación

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente.

C++: es un lenguaje de programación diseñado a mediados de los años 80 por Bjarne Troustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido. Posteriormente, se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que C++ es un lenguaje de programación multiparadigma.

Las principales características del lenguaje C++ son:

- Tiene un conjunto completo de instrucciones de control.
- Permite la agrupación de instrucciones.
- Incluye el concepto de puntero (variable que contiene la dirección de otra variable).
- Los argumentos de las funciones se transfieren por su valor.
- Las entradas/salidas no forman parte del lenguaje, sino que se proporciona a través de una biblioteca de funciones.
- Permite la separación de un programa en módulos que admiten compilación independiente.
- Programación de bajo nivel (nivel bit).

C++ es un derivado del mítico lenguaje C. Este lenguaje apareció en la década de los 70 de la mano de Dennis Ritchie para la programación en sistemas operativos Unix (el mejor ejemplo actual de un sistema operativo Unix es GNU/Linux), el cual surgió como un lenguaje generalista recomendado sobre todo para programadores ya expertos, ya que no llevaba implementadas muchas funciones que hacen a un lenguaje más comprensible. Sin embargo, aunque esto en un principio puede convertirse en un problema, en la práctica es su mayor virtud, ya que permite al programador un mayor control sobre lo que está haciendo.

Algunas ventajas del uso de este lenguaje son: incorpora soporte para el paradigma orientado a objetos, es muy potente en lo que se refiere a creación de sistemas complejos, se caracteriza por su robustez,

incorpora el lenguaje C lo que permite el uso de sus instrucciones, librerías y características, gracias a sus herramientas otorga un control mucho más amplio sobre las operaciones que se desean realizar, permite administrar de forma manual y automática la memoria del sistema por lo que no es obligatorio el uso de recolectores de basura. En esta última década se han desarrollado un sin número de aplicaciones de escritorio, web y para todo tipo de dispositivos electrónicos con el uso de este lenguaje.

***Bourne Again Shell* (Bash):** Bash es un intérprete de comandos del proyecto GNU. Es un lenguaje interpretado de programación, que facilita el trabajo a los administradores, para realizar la mayoría de las actividades en el sistema, por lo que es más que una simple consola. Su mayor utilización se ve en los sistemas Unix. Este intérprete incorpora algunas características que lo convierte en uno de los intérpretes de comandos más utilizados a nivel mundial.

Como lenguaje de alto nivel proporciona variables, control de flujo, funciones, control de procesos, redirección de entrada/salida y un lenguaje de orden para escribir programas por *script*. Incluye características de otros intérpretes de comandos como es el caso de la sintaxis Bourne Shell, el redireccionamiento, comandos y variables de *Korn shell* y la edición de comandos de *C shell,* los cuales forman parte de la familia de intérpretes de comandos de la Unix. Siendo considerado como el intérprete de comandos más extendido en Linux, posee una sintaxis familiar y asequible para programas externos, propicio para sistemas de autocompilado. El soporte del mismo para funciones permite al sistema acoger un diseño modular fácil de entender, sacando 38 provechos del mismo permitiendo a los mantenedores de paquetes y a los desarrolladores reconfigurarlo inmediatamente (26).

Las principales características del intérprete bash son:

- Ejecución síncrona de órdenes (una tras otra) o asíncrona (en paralelo).
- Distintos tipos de redirecciones de entradas y salidas para el control y filtrado de la información.
- Control del entorno de los procesos.
- Ejecución de mandatos interactiva y desatendida, aceptando entradas desde teclado o desde ficheros.
- Proporciona una serie de órdenes internas para la manipulación directa del intérprete y su entorno de operación.
- Un lenguaje de programación de alto nivel, que incluye distintos tipos de variables, operadores, matrices, estructuras de control de flujo, entrecomillado, sustitución de valores y funciones.

- Control de trabajos en primer y segundo plano.
- Edición del histórico de mandatos ejecutados.
- Posibilidad de usar una *shell*[20] para controlar el entorno del usuario.

1.4.5 *Debmirror*.

Herramienta utilizada mediante líneas de comandos que permite la creación parcial o total de un repositorio Debian o Ubuntu. El mismo descarga un *mirror* o espejo local de todos los paquetes del servidor principal admitiendo cualquier combinación de arquitecturas, distribuciones y secciones. Los archivos son transferidos por ftp, http, hftp[21] o rsync[22] y es totalmente compatible con todos los repositorios oficiales (27).

1.4.6 Entorno de desarrollo.

Un entorno de desarrollo integrado o *Integrated Development Environment* (IDE) es un programa que le brinda a los desarrolladores de *software* un conjunto de herramientas que facilitan su desempeño laboral y proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación existentes, tales como Java, C++, C#, Python, etc. Ofrecen por lo general un editor de textos especializado como herramienta central, así como navegadores de archivos, asistentes de compilación, navegadores de objetos, lectores de documentación, entre otros (28).

Qt Creator: es un Entorno Integrado de Desarrollo (IDE) creado por *Trolltech*, es multiplataforma y fue diseñado para desarrollar usando lenguajes como C/C++ y Python con la integración del *framework* Qt, para lograr una agilización y simplificación en el desarrollo de aplicaciones que pueden ser para escritorio, web o para móviles. En el año 2008, Nokia compra Qt a Trolltech y luego en el 2011 tras su alianza con Microsoft decide vender la licencia comercial de Qt a Digia. Qt Creator no quiere ser un reemplazo de Eclipse ni Visual Studio, sino un IDE ligero pensado especialmente para el desarrollo en múltiples plataformas: Windows, Linux (desde la versión 2.6) y Mac OSX (desde 10.4 en adelante).

[20] Término usado en informática para referirse a un intérprete de comandos, el cual consiste en la interfaz de usuario tradicional de los sistemas operativos basados en Unix y similares como GNU/Linux.

[21] Protocolo para acceder a recursos a través de ftp por el proxy http.

[22] Aplicación libre para sistemas de tipo Unix y Microsoft Windows que ofrece transmisión eficiente de datos incrementales, que opera también con datos comprimidos y cifrados.

Qt Creator se centraliza en proporcionar características que ayudan a los nuevos usuarios de Qt a aprender y comenzar a desarrollar rápidamente, también aumenta la productividad de los desarrolladores con experiencia en Qt.

Algunas de las ventajas que presenta son las siguientes:

- Editor de código con soporte para C++*, QML y ECMAscript*.
- Herramientas para la rápida navegación del código.
- Resaltado de sintaxis y auto-completado de código.
- Control estático de código y estilo a medida que se escribe.
- Soporte para re-factorizar código.
- Ayuda sensitiva al contexto.
- Plegado de código (*code folding*).
- Paréntesis coincidentes y modos de selección.

El depurador visual (*visual debugger*) para C++, es consciente de la estructura de muchas clases de Qt, lo que aumenta la capacidad de mostrar los datos de Qt con claridad. Además, Qt Creator muestra la información en bruto procedente de GDB[23] de una manera clara y concisa.

- Interrupción de la ejecución del programa.
- Ejecución línea por línea o instrucción a instrucción.
- Puntos de interrupción (*breakpoints*).

[23] GDB fue escrito por Richard Stallman en 1986. **GDB** es software libre distribuido bajo la licencia GPL. GDB ofrece la posibilidad de trazar y modificar la ejecución de un programa. El usuario puede controlar y alterar los valores de las variables internas del programa.

- Examinar el contenido de llamadas a la pila (*stack*), los observadores y de la variables locales y globales.

Presenta un entorno integrado para la creación y diseño de *forms* (ventanas) para proyectos C++ que permite diseñar rápidamente *widgets* y diálogos usando los mismos *widgets* que se usarán en su aplicación. Los *forms* son totalmente funcionales y pueden ser pre-visualizados inmediatamente para asegurarse de que se verá y sentirá exactamente como lo pensaron los desarrolladores.

1.5 Conclusiones del capítulo.

En este capítulo se abordaron todos los elementos teóricos que sustentan las posibles soluciones al problema planteado.

El análisis de los referentes teóricos metodológicos facilitó la comprensión de diferentes temas asociados al problema de la investigación. Los conceptos asociados al dominio de la investigación aportaron claridad en la división del problema de la investigación en pequeños problemas menos complejos.

La selección de la metodología de desarrollo OpenUp permitió estructurar, planificar y controlar el proceso de desarrollo de la propuesta de solución y garantizar la calidad de dicho proceso. Finalmente, el estudio realizado de las tecnologías y lenguajes de programación, permitió determinar las bases tecnológicas necesarias para el desarrollo de la aplicación.

Capítulo 2: "Análisis y diseño de la aplicación"

Introducción

La creación de sistemas informáticos es un proceso en el que se desarrollan artefactos que, luego de ser integrados, posibilitan responder a las necesidades del cliente. En este proceso se identifican varias etapas, que van desde la declaración del problema y los requisitos del sistema, hasta las pruebas y la liberación del mismo. En el presente capítulo se define la propuesta del sistema a desarrollar, con el propósito de satisfacer el objetivo de la investigación. Para optimizar la comprensión de la solución propuesta se realiza el análisis y diseño utilizando la metodología ágil OpenUp, donde describe el modelo conceptual y los principales procesos del sistema mediante la especificación de casos de uso. Como parte del diseño de la solución se desarrollan los diagramas de clases y de secuencia, que constituyen los artefactos principales generados para esta etapa.

2.1 Propuesta de solución.

La compresión del repositorio para la distribución cubana GNU/Linux Nova permitirá disminuir el tiempo de descarga de dicho repositorio facilitando así el tráfico en la red, además de poder satisfacer las necesidades de los clientes y disminuir el almacenamiento necesario en los repositorios espejos.

Para la solución se desarrolló dos script[24] programados en bash, los cuales permitieron darle respuesta a dicho problema. Dicho proceso se describe a continuación:

[24] Fichero de texto conteniendo comandos externos e internos, que se ejecutan línea por línea.

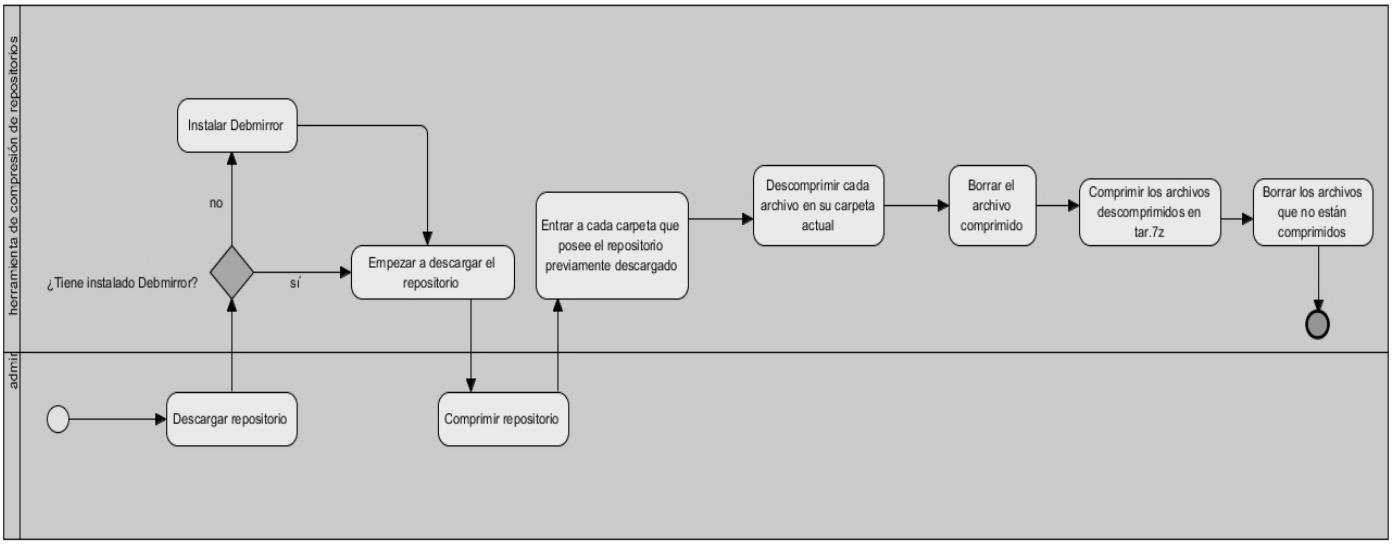


Figura 4: Proceso para comprimir repositorio de Nova.

2.2 Modelo del dominio.

El Modelo de Dominio o Modelo Conceptual es una representación visual de los principales conceptos u objetos del mundo real, significativos para un problema o área de interés. Este es de gran ayuda para desarrolladores y usuarios, de esta forma se utiliza un vocabulario común y pueden entender el contexto en que se enmarca el sistema (29).

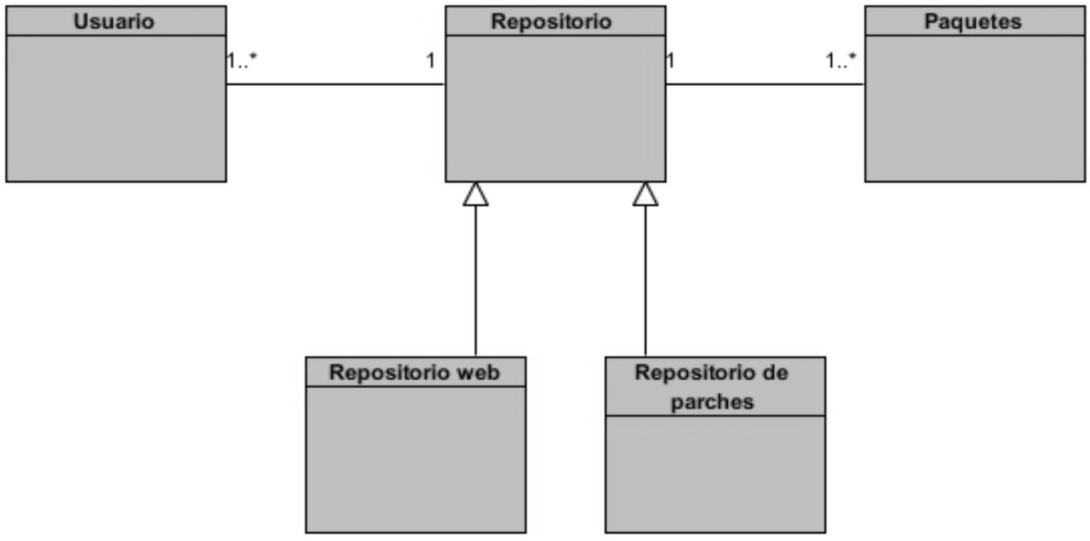


Figura 5: Modelo de dominio.

2.2.1 Descripción de las clases del Modelo de Dominio.

Tabla 4: Descripción de los objetos que intervienen en el proceso de implementación.

Objeto o relación	Descripción
Usuario	Persona que interactúa con la aplicación encargada de la compresión del repositorio para la distribución cubana GNU-Linux Nova.
Repositorio	Conjunto de paquetes de *software* disponibles para una determinada distribución almacenados de forma centralizada.
Repositorio web	Repositorio localizado en la web.
Repositorio de parches	Repositorio minimizado que contiene los paquetes de *software* fundamentales.

2.3 Modelado del Sistema.

Con el objetivo de realizar el modelado del sistema de una forma eficiente y que cumpla con las expectativas de los clientes, se hace uso de la metodología OpenUp y se aplican las vistas de casos de uso, vista lógica y vista de desarrollo. Por otra parte la metodología OpenUp define en su primera fase la identificación de los requisitos que debe cumplir el sistema a desarrollar. A continuación se muestran los requisitos identificados mediante las técnicas de captura de requisitos: Tormenta de ideas.

2.3.1 Requisitos del sistema.

Un requisito es una condición o capacidad que debe tener un sistema para satisfacer las necesidades de un cliente, con el fin de resolver un problema planteado por el mismo en forma de un documento formal (30). Estos se dividen en Requisitos Funcionales (RF) y Requisitos No Funcionales (RNF).

2.3.1.1 Requisitos funcionales.

Los requisitos funcionales definen las funciones que es capaz de realizar el sistema, es decir, describen las funcionalidades o los servicios que se espera que este provea (30). A continuación, los requisitos funcionales del sistema:

RF_1 Obtener lista de paquetes del repositorio.

RF_2 Obtener última versión de los paquetes.

RF_3 Descargar código fuente del repositorio.

RF_4 Permitir la comprensión de archivos en formato tar.7z.

RF_5 Permitir la descomprensión de archivos en formato tar.7z.

RF_6 Reducir el tamaño del repositorio de la distribución cubana GNU-Linux Nova.

2.3.2 Diagrama de casos de uso del sistema.

El diagrama de casos de uso del sistema ayuda a comprender gráficamente los procesos del sistema y su interacción con los actores.

Tabla 5: Actor del sistema y acción que realiza.

Actores	Justificación
Usuario	Persona que accede al repositorio para descargar los paquetes de una determinada distribución.

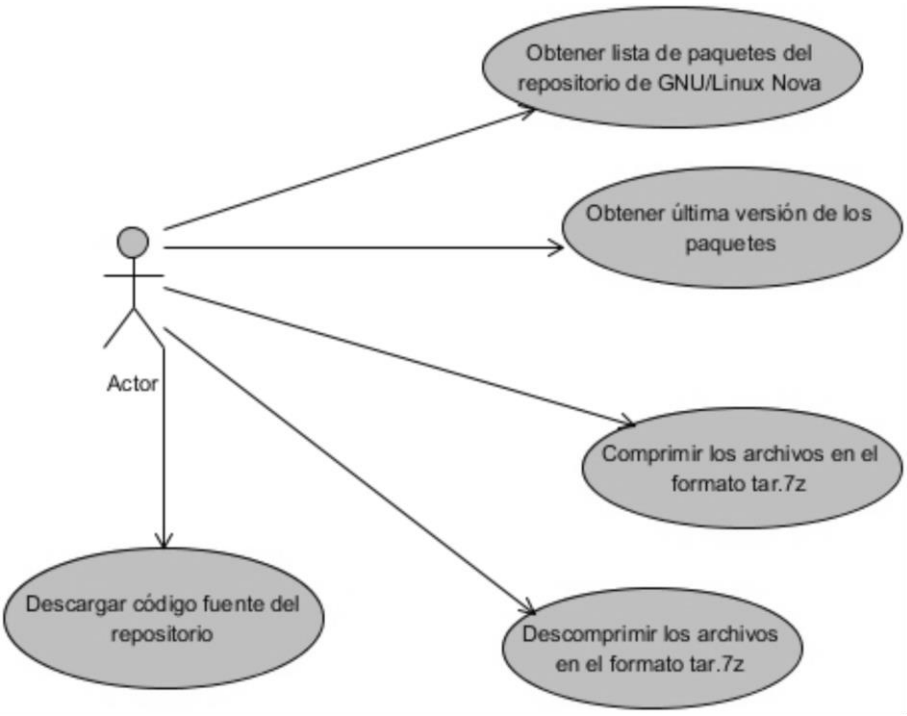


Figura 6: Diagrama de Casos de Uso.

Tabla 6: Descripción del Caso de Uso Obtener lista de paquetes del repositorio GNU/Linux Nova.

Caso de Uso #1	Obtener lista de paquetes del repositorio GNU/Linux Nova.
Actores	Usuario.
Resumen	El caso de uso inicia cuando el usuario se conecta al repositorio para descargar los paquetes disponibles de la distribución cubana GNU/Linux Nova y finaliza cuando se

	obtiene el listado con todos los paquetes disponibles de la distribución cubana de GNU/Linux Nova.
Referencias	RF_1
Complejidad	Media
Prioridad	Alta
Precondiciones	El usuario necesita obtener la lista de paquetes de *software* de la distribución cubana de GNU/Linux Nova.
Postcondiciones	Estar conectado al repositorio.

Flujo de eventos

Flujo básico "Descargar paquetes"

	Actor	**Sistema**
1	Solicita obtener el listado de paquetes de *software* del repositorio de la distribución cubana de GNU/Linux Nova.	
2		El sistema busca los paquetes de *software* que estén disponibles para el usuario.
3		Se obtiene el listado de paquetes de *software* disponibles.
4		Termina el caso de uso.

Flujos alternos

1."No hay paquetes de *software* disponibles".

	Actor	**Sistema**
1	Solicita obtener el listado de paquetes de *software* del repositorio de la distribución cubana GNU/Linux Nova.	
2		El sistema busca los paquetes de *software* que estén disponibles para el usuario.
3		Emite una notificación sobre el suceso.
4		Termina el caso de uso.

Tabla 7: Descripción del Caso de Uso Obtener última versión de los paquetes del repositorio GNU/Linux Nova.

Caso de Uso #2	Obtener última versión de los paquetes del repositorio GNU/Linux Nova.	
Actores	Usuario.	
Resumen	El caso de uso inicia cuando el usuario solicita obtener la última versión de los paquetes de la distribución cubana de GNU/Linux Nova y finaliza cuando se obtiene la versión de los paquetes disponibles de la distribución cubana de GNU/Linux Nova.	
Referencias	RF_2	
Complejidad	Media	
Prioridad	Alta	
Precondiciones	El usuario necesita obtener la de última versión de los paquetes de *software* de la distribución cubana de GNU/Linux Nova.	
Postcondiciones		
Flujo de eventos		
Flujo básico "Obtener última versión de los paquetes"		
	Actor	**Sistema**
1	Solicita obtener la última versión de los paquetes de *software* del repositorio de la distribución cubana de GNU/Linux Nova.	
2		El sistema busca la última versión de los paquetes de *software*.
3		Se obtiene la última versión de los paquetes de *software*.
4		Termina el caso de uso.
Flujos alternos		
1."No se obtiene la última versión de los paquetes".		
	Actor	**Sistema**
1	Solicita obtener la última versión de los paquetes de *software* del repositorio de la distribución cubana de GNU/Linux Nova.	
2		El sistema busca la última versión de los paquetes de *software*.
3		Emite una notificación sobre el suceso.

4		Termina el caso de uso.

Tabla 8: Descripción del Caso de Uso Descargar código fuente del repositorio.

Caso de Uso #3	Descargar código fuente del repositorio.	
Actores	Usuario.	
Resumen	El caso de uso inicia cuando el usuario desea descargar el código fuente del repositorio y finaliza cuando se descarga el código fuente.	
Referencias	RF_3	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	El usuario necesita descargar el código fuente de los repositorios.	
Postcondiciones		
Flujo de eventos		
Flujo básico "Descargar código fuente"		
	Actor	**Sistema**
1	Solicita descargar el código fuente del repositorio.	
2		El sistema busca el código fuente del repositorio.
3		Se descarga el código fuente del repositorio.
4		Termina el caso de uso.
Flujos alternos		
1."No se puede descargar el código fuente".		
	Actor	**Sistema**
1	Solicita descargar el código fuente del repositorio.	
2		El sistema busca el código fuente del repositorio.

3		Emite una notificación sobre el suceso.
4		Termina el caso de uso.

Tabla 9: Descripción del Caso de Uso Comprimir los archivos en tar.7z.

Caso de Uso #4	Comprimir los archivos en tar.7z.	
Actores	Usuario.	
Resumen	El caso de uso inicia cuando el usuario desea comprimir los archivos descargados del repositorio en el formato tar.7z y finaliza cuando se comprimen los archivos en el formato tar.7z.	
Referencias	RF_4	
Complejidad	Media	
Prioridad	Alta	
Precondiciones	El usuario necesita comprimir los archivos descargados del repositorio.	
Postcondiciones		
Flujo de eventos		
Flujo básico “Comprimir archivos”		
	Actor	**Sistema**
1	Solicita comprimir los archivos en tar.7z.	
2		El sistema comprime los archivos en tar.7z.
3		Termina el caso de uso.
Flujos alternos		
1.”No se puede comprimir en tar.7z”.		
	Actor	**Sistema**
1	Solicita comprimir los archivos en tar.7z.	
2		Emite una notificación sobre el suceso.
3		Termina el caso de uso.

Tabla 10: Descripción del Caso de Uso Descomprimir los archivos en tar.7z.

Caso de Uso #5	Descomprimir los archivos en tar.7z.	
Actores	Usuario.	
Resumen	El caso de uso inicia cuando el usuario desea descomprimir los archivos descargados comprimidos del repositorio en el formato tar.7z y finaliza cuando se descomprimen los archivos.	
Referencias	RF_5	
Complejidad	Media	
Prioridad	Alta	
Precondiciones	El usuario necesita descomprimir los archivos descargados del repositorio.	
Postcondiciones		
Flujo de eventos		
Flujo básico "Descomprimir archivos"		
	Actor	**Sistema**
1	Solicita descomprimir los archivos en tar.7z.	
2		El sistema descomprime los archivos en tar.7z.
3		Termina el caso de uso.
Flujos alternos		
1."No se puede descomprimir en tar.7z".		
	Actor	**Sistema**
1	Solicita descomprimir los archivos en tar.7z.	
2		Emite una notificación sobre el suceso.
3		Termina el caso de uso.

2.4 Definición de la arquitectura.

La arquitectura de *software* es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un *software*, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores del *software* compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones

de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del *software* (31).

Para el desarrollo y organización estructural de la aplicación se propone como arquitectura base, una arquitectura 3 capas, constituyendo una distribución jerárquica de responsabilidades para proporcionar una división de los problemas a resolver. La arquitectura propuesta para el desarrollo de la aplicación de compresión del repositorio de la distribución cubana GNU-Linux Nova identifica como capas:

- **Capa de Vista:** están contenidos los componentes visuales con los que el usuario interactúa.
- **Capa Controlador:** la capa controladora gestiona las peticiones de los usuarios. Es responsable de responder la información solicitada con la ayuda tanto del proceso como de la vista.
- **Capa de Procesos:** se encuentran las implementaciones concretas de cada uno de los procedimientos necesarios para desarrollar las estrategias de compresión del repositorio.

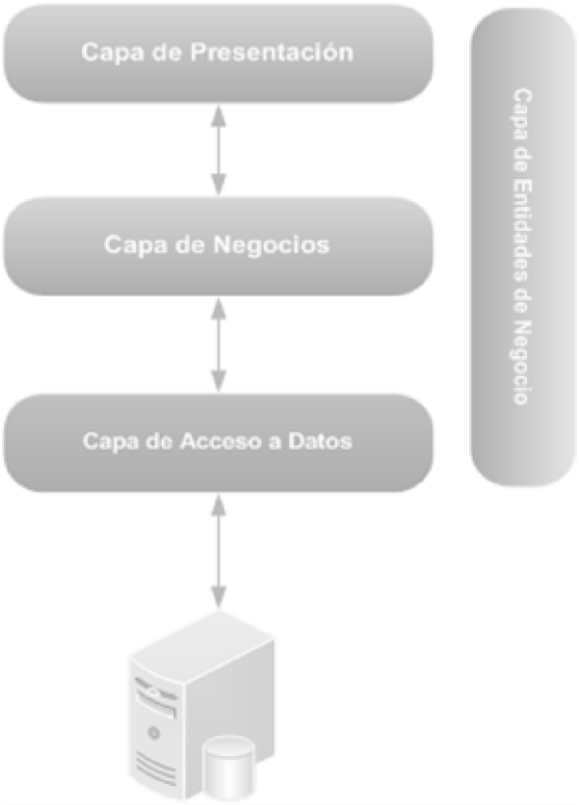


Figura 7: Arquitectura 3 capas.

2.5 Patrones de diseño.

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de *software*, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (32).

- **Patrones GRASP:** los Patrones de Principios Generales para Asignar Responsabilidades (GRASP por sus siglas en inglés) describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones.
- **Experto en información:** las responsabilidades deben ser asignadas a las clases que poseen la información para realizar dicha responsabilidad.
- **Creador:** asignarle a una clase la responsabilidad de crear una instancia de otra. Dentro del sistema este patrón se evidencia en las acciones de los controladores, las cuales crean objetos del modelo o los formularios que representan las entidades.
- **Alta cohesión:** una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Significa que las clases del sistema tienen asignadas solo las responsabilidades que les corresponde y mantienen una estrecha relación con el resto de las clases.
- **Bajo acoplamiento:** determina el nivel de dependencia de una clase con respecto a otras. Una clase con bajo acoplamiento no depende de muchas otras.
- **Controlador:** es el encargado de asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones. Se evidencia el uso de este patrón en la aplicación, ya que para cada petición o evento que se genere en el mismo, existe un controlador con la responsabilidad de obtenerla y devolver una respuesta. La respuesta puede ser mostrar una vista, ejecutar un método, devolver un mensaje, etc (32).

2.6 Diagrama de clases

Los diagramas de clases sirven para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contención. Los diagramas de clases están compuestos por atributos y métodos, así como las relaciones estáticas que existen entre ellas.

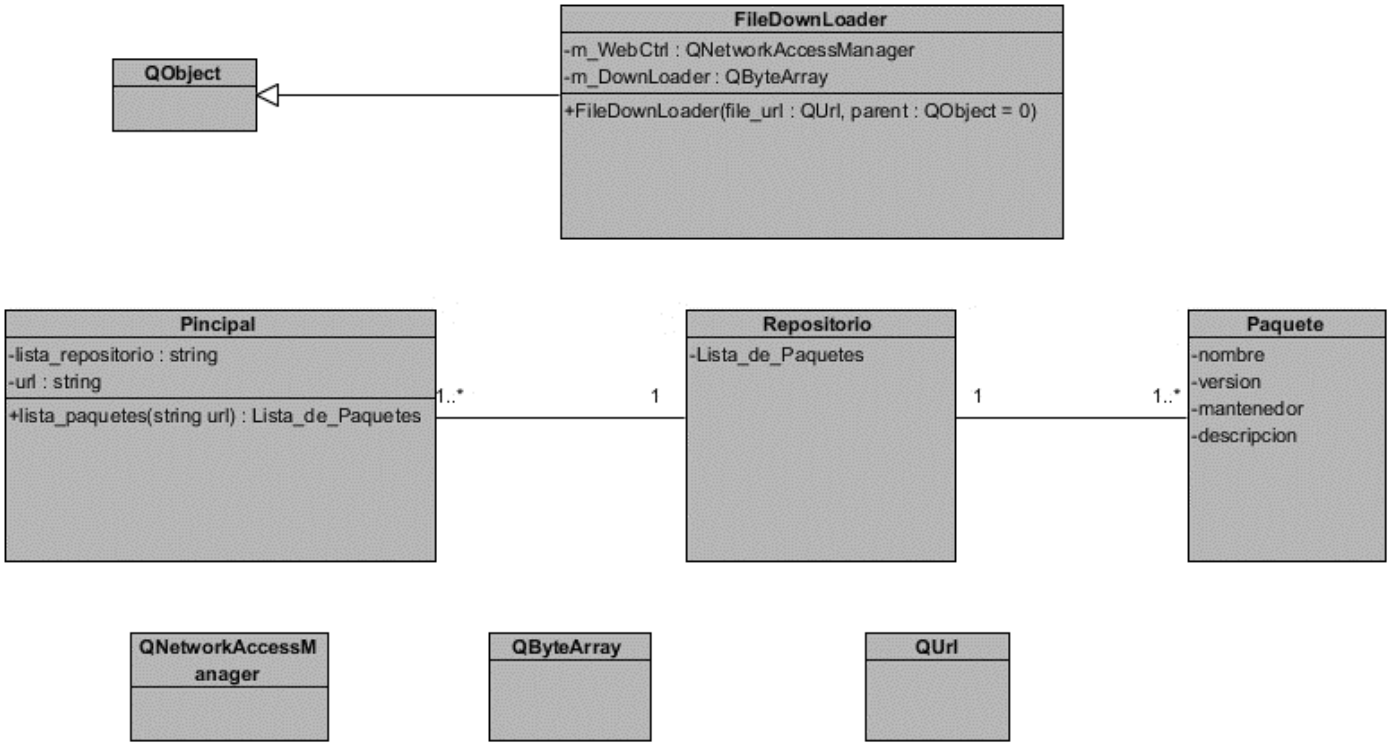


Figura 8: Diagrama de clases.

2.7 Conclusiones del capítulo.

En este capítulo fueron descritos los procesos que intervienen en el negocio y se arribó a las siguientes conclusiones.

Se realizó el modelado del sistema teniendo como guía la metodología OpenUp. Se generaron además los artefactos necesarios para la implementación de la solución, lográndose el desarrollo de las funcionalidades que dan cumplimiento a los requisitos del sistema y fueron analizados los patrones de diseño asociados a la arquitectura propuesta, los cuales darán mayor independencia a las clases y facilitarán la implementación.

Al concluir el presente capítulo, se han creado las condiciones para efectuar la implementación de la herramienta de compresión de repositorios para la distribución cubana de GNU\Linux Nova.

Capítulo 3: “Implementación y pruebas”

Introducción

La fase de implementación en el desarrollo de un producto de *software*, es el mecanismo donde se ponen en práctica todas las descripciones y arquitecturas propuestas en las fases de análisis y diseño, es el complemento del trabajo de las fases que lo preceden dentro del proceso de desarrollo de *software*. La implementación ofrece una materialización precisa de los requisitos. Una de las últimas fases del ciclo de vida antes de entregar un *software* para su explotación es la fase de pruebas, cuyo objetivo es comprobar si este cumple sus requisitos. Dentro de ella pueden desarrollarse varios tipos de pruebas en función de los objetivos de las mismas.

3.1 Código fuente

Para obtener una versión funcional de la aplicación se deben implementar los componentes que se han definido, como resultado se obtienen archivos que contienen el código fuente de la aplicación. El código fuente de un *software* es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está escrito su funcionamiento. Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones (33).

3.1.1 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

Para facilitar el entendimiento del código y fijar un modelo a seguir, se establecieron estándares de codificación. A continuación, se muestran algunos de estos estándares para el lenguaje C++ utilizados en el *framework* Qt.

Nombres de clases y métodos

Para la definición de las clases y métodos en el código de la aplicación se utilizó el estándar *CamelCase*. Este es un estilo de escritura que se aplica a frases o palabras compuestas. Existen dos tipos de estándares de *CamelCase* (34):

- ***UpperCamelCase*:** cuando la primera letra de cada una de las palabras es mayúscula. Ejemplo: EjemploDeUpper*CamelCase*.
- ***lowerCamelCase*:** igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: ejemploDeLower*CamelCase*.

En la definición del nombre de las clases fue utilizado *UpperCamelCase* y en la nomenclatura de los métodos lower*CamelCase*:

```
#include "control.h"
#include "mainwindow.h"
#include "QProcess"

Control::Control(QObject *parent) : QObject(parent)
{
    dl = new FileDownloader();
    connect(dl,SIGNAL(downloaded()),this,SLOT(WriteFile()));
    connect(dl,SIGNAL(UpdatePorcess(int)),this,SLOT(UpdateProcessSLOT(int)));
    connect(this,SIGNAL(NextSolicitude()),this,SLOT(Begin()));
    connect(dl,SIGNAL(Exception(QString)),this,SLOT(ShowException(QString)));
    task=0;
}

Control::~Control()
{

}

void control::Begin()
{
    if(!listado_descargas.empty())
    {
     QUrl temp = listado_descargas.front();
     listado_descargas.pop_front();
     emit UpdateDescriptionSIGNAL(QString("Iniciando descarga: %1").arg(temp.path()));
     emit UpdateProcessSIGNAL(((float)(task-listado_descargas.size())/(float)task)*100.0f);
     dl->setUrl(temp);
     dl->start();
    }

}

void control::ShowException(QString e)
{
    QMessageBox msgBox;
    msgBox.setText("Error:"+e);
    msgBox.exec();
    emit NextSolicitude();
}
```

Figura 9: Uso del estándar de codificación *CamelCase*.

Estructura

1. El código debe usar cuatro espacios para la identación en vez de usar el tabulado. Esto minimiza problemas con otras herramientas de desarrollo.
2. Las líneas podrían tener 80 caracteres o menos, evitando tener más de 120 caracteres.
3. Las llaves de apertura deben ir en la siguiente línea y la llave de cierre debe ir en la siguiente línea después del cuerpo.

4. Las llaves de apertura en las estructuras de control debe ir en la misma línea y las llaves de cierre deben de ir después del cuerpo.
5. Los paréntesis en las estructuras de control no deben usar espacios antes o después.
6. Añadir un solo espacio después de cada limitador de coma.
7. Añadir un solo espacio alrededor de los operadores (==, &&,…).
8. Usa llaves para indicar el control de la estructura sin tener en cuenta el número de declaraciones que el grupo pueda contener.
9. Definir una clase por fichero.
10. Declarar las propiedades de clase antes que los propios métodos de clase.

3.2 Validación del sistema

Una vez terminada la implementación del producto que se requiere, es necesario realizarle pruebas con el objetivo de detectar errores en la aplicación y la documentación; este proceso resulta de gran importancia ya que da una medida de la calidad del mismo siempre que se lleve a cabo de la forma correcta. A continuación se muestran las pruebas realizadas al sistema y los resultados obtenidos por cada una.

3.2.1 Pruebas funcionales o de caja negra

Las pruebas de caja negra, también denominadas pruebas funcionales se centran en los requisitos funcionales del *software*. O sea, la prueba de caja negra permite al ingeniero del *software* obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (35).

Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro. Las pruebas funcionales que se realizarán a la solución, estarán enfocadas o dirigidas a los casos de uso del sistema para verificar su correcto funcionamiento. En este tipo de pruebas se ejecutarán los distintos servicios prestados con datos correctos e incorrectos. En caso de que los datos sean incorrectos se verificará que los mensajes de error sean los deseados y en el caso opuesto que los resultados sean los esperados.

A continuación, se presentan los casos de prueba correspondientes al caso de uso "Obtener lista de paquetes del repositorio GNU-Linux Nova." y "Comprimir archivos en el formato tar.7z". El resto de los casos de prueba pueden ser consultados en el Anexo 1.

Tabla 11: Caso de prueba para el Caso de Uso "Obtener lista de paquetes del repositorio GNU-Linux Nova".

Sección "Descargar paquetes"			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Obtener el listado de paquetes de *software*.	Se obtiene el listado de paquetes de *software* correctamente.	Descarga el listado de paquetes de *software* disponibles.	1. Se selecciona el botón "Descargar".
EC 1.2 No hay paquetes de *software* disponibles.	No se obtienen los paquetes de *software*.	Muestra un mensaje informando del suceso.	1. Se selecciona el botón "Descargar". 2. Se muestra el mensaje informando el suceso.

Tabla 12: Caso de prueba para el Caso de Uso "Comprimir archivos en el formato tar.7z".

Sección "Comprimir archivos"			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Comprimir los archivos en tar.7z.	Se comprimen los archivos correctamente.	Comprime el repositorio en el formato tar.7z.	1. Se selecciona el botón "Comprimir".
EC 1.2 No se puede comprimir en el formato tar.7z.	No se comprime el repositorio.	Muestra un mensaje informando del suceso.	1. Se selecciona el botón "Comprimir". 2. Se muestra el mensaje informando el suceso

3.2.2 Pruebas de Aceptación

Los clientes escriben las pruebas funcionales para cada caso de uso que deba validarse. Un caso de uso no es aceptado hasta que haya pasado su prueba de aceptación. Las pruebas de aceptación representan algún tipo de resultado por parte del sistema. Los clientes son los responsables de verificar la exactitud de

estas pruebas y de revisar los resultados para poder así priorizar las que fracasaron. Esto significa que en cada iteración se deben realizar nuevas pruebas de aceptación. Las pruebas de aceptación en OpenUp permiten verificar que las funcionalidades de cada iteración se cumplan correctamente. El probador es el responsable de ayudar al cliente a seleccionar y escribir las pruebas de aceptación para cada historia de usuario. Tiene la responsabilidad de ayudar al cliente a tomar las decisiones correctas sobre qué significa la calidad para su proyecto.

Tabla 13: Prueba de aceptación # 1.

Caso de prueba de aceptación	
Código: CU1_P1	**Caso de Uso :** 1
Nombre: Obtener lista de paquetes del repositorio GNU-Linux Nova.	
Descripción: Prueba para la funcionalidad Obtener lista de paquetes del repositorio GNU-Linux Nova.	
Condiciones de ejecución: Estar conectado al repositorio.	
Pasos de ejecución: El usuario necesita obtener la lista de paquetes de *software* de la distribución GNU-Linux Nova.	
Resultados esperados: El usuario obtiene la lista de paquetes de *software*.	

Tabla 14: Prueba de aceptación # 2.

Caso de prueba de aceptación	
Código: CU2_P2	**Caso de Uso: 2**
Nombre: Obtener última versión de los paquetes del repositorio GNU-Linux Nova.	
Descripción: Prueba para la funcionalidad Obtener última versión de los paquetes del repositorio GNU/Linux Nova.	
Condiciones de ejecución: Estar conectado al repositorio.	
Pasos de ejecución: El usuario necesita obtener la última versión de los paquetes de *software*.	

Resultados esperados: El usuario obtiene la última versión de los paquetes de *software*.

Tabla 15: Prueba de aceptación # 3.

Caso de prueba de aceptación	
Código: CU3_P3	**Caso de Uso: 3**
Nombre: Descargar código fuente del repositorio.	
Descripción: Prueba para la funcionalidad Descargar código fuente del repositorio.	
Condiciones de ejecución: Estar conectado al repositorio.	
Pasos de ejecución: El usuario presiona el botón “Descargar” y empieza a descargar el código fuente del repositorio.	
Resultados esperados: El usuario descarga el código fuente del repositorio.	

Tabla 16: Prueba de aceptación # 4.

Caso de prueba de aceptación	
Código: CU4_P4	**Caso de Uso: 4**
Nombre: Comprimir los archivos en el formato tar.7z.	
Descripción: Prueba para la funcionalidad Comprimir los archivos en el formato tar.7z.	
Condiciones de ejecución: Estar conectado al repositorio.	
Pasos de ejecución: El usuario presiona el botón “Comprimir” y empieza a comprimir el repositorio previamente descargado.	
Resultados esperados: El usuario comprime el repositorio en el formato tar.7z.	

3.2.3 Validación del proceso de compilación

El proceso de compilación que se verifica, cumple con la condición de que durante el trabajo para construir paquetes, se utilice el algoritmo de compresión tar.7z, tanto para los de código fuente como para los archivos internos de los paquetes .deb.

Para comprobar que se disminuya el tamaño del repositorio, se realiza la siguiente comparación de la tabla 17 con una muestra de un repositorio de 5 paquetes. Dicha verificación se hace con los tamaños del repositorio creado con los paquetes compilados con métodos de tar.7z y el repositorio sin modificaciones.

Tabla 17: Comparación de tamaños de paquetes compilados con tar.7z.

Paquetes	Paquete compilado sin compresión tar.7z (MB)	Paquete compilado con compresión tar.7z (MB)
linux_3.13.0	513.6	80.9
firefox_38.0+build	872.5	162.2
openjdk-6-dbg_6b35-1.13.7-1ubuntu0.14.04.1_i386	555.2	98.5
thunderbird_31.7.0+build1	595.2	141
Libvirt-1.2.2	144.2	10.9
Total	2680.7	493.5

Tales resultados se muestran en la siguiente gráfica:

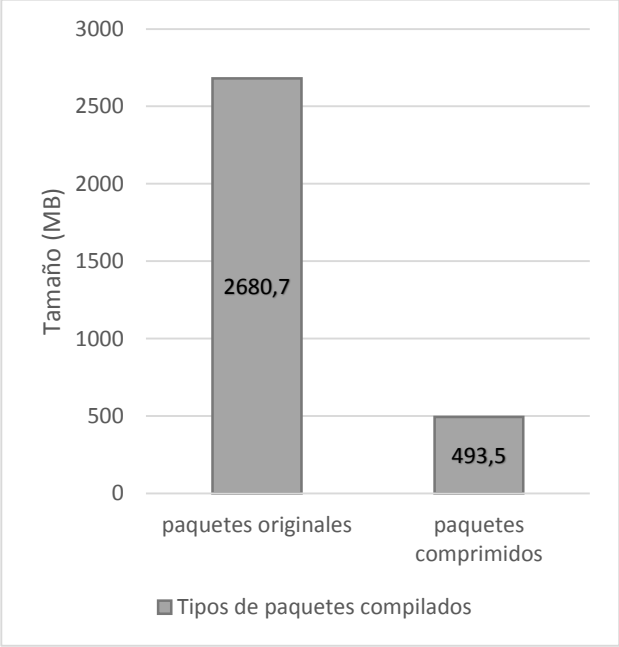


Figura 10: Tamaños de repositorio original y repositorio comprimido internamente en tar.7z.

Dicho repositorio de muestra, comprimido con el nuevo método de compresión, supone un ahorro del 19% del espacio en el servidor, pues representa el 71% del tamaño total.

Se realizó también la prueba para la rama principal del repositorio actual de nova que consta de un tamaño de 10 Gb:

Tabla 18: Evolución del ahorro de tamaños aplicando propuestas en el repositorio.

Tamaños de descargas(Gb)	Propuesta aplicada	Resultado de tamaño de descarga (Gb)

10	Aplicar compresión a paquetes en tar.7z.	5

Tal resultado se detalla en la siguiente gráfica:

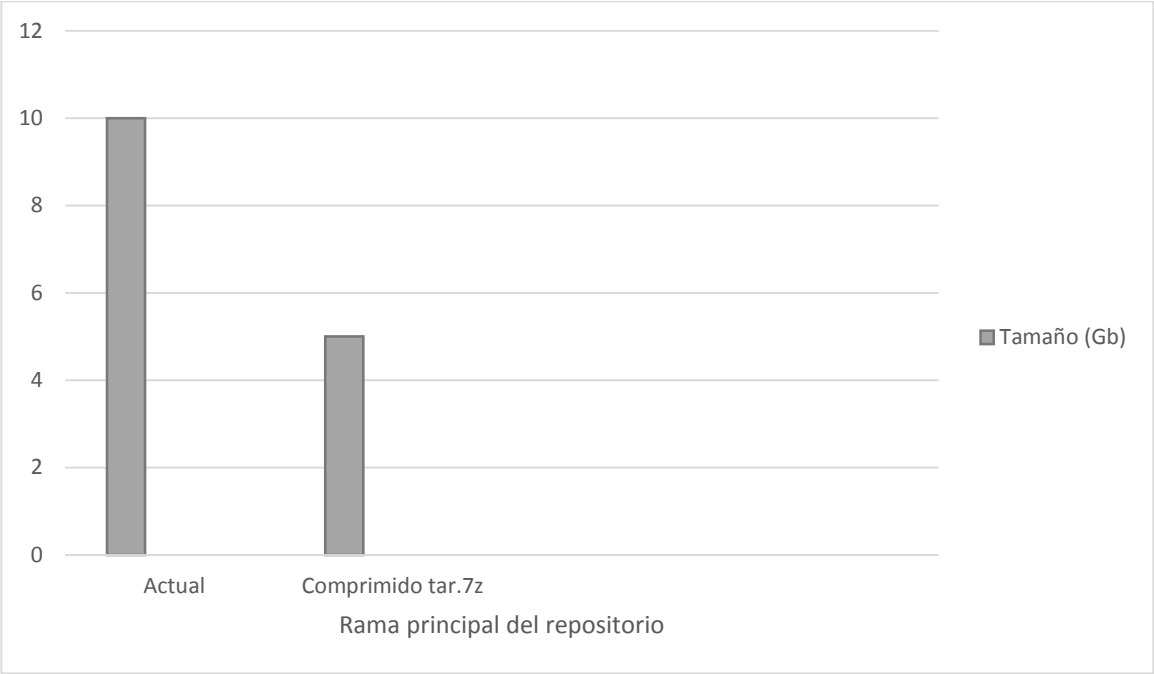


Figura 11: Tamaño y tiempo de evolución de reducción en volumen.

Según los resultados obtenidos, se puede confirmar que después de haber compilados todos los paquetes del repositorio, se va a tener un repositorio reducido el cual representa el 50% del repositorio actual aproximadamente, lo cual va a incidir en que las descargas del mismo sean más rápidas, y así poder facilitar el tráfico en la red y poder satisfacer las necesidades de los clientes.

3.3 Conclusiones del capítulo

Se definió la notación estándar *CamelCase* para establecer una uniformidad en la etapa de implementación del sistema. Las pruebas realizadas permitieron identificar algunos defectos, los que fueron corregidos y permitieron aumentar la calidad final de la solución. Tras el flujo de implementación y prueba, el sistema quedó desarrollado. Por tanto, la culminación del presente capítulo representa la garantía, calidad y satisfacción conjunta del cliente y el equipo de desarrollo con el producto final.

Conclusiones

Una vez realizada la fundamentación teórica que respaldó este trabajo, definidas las características del sistema, efectuada la implementación y validación del mismo, se obtuvieron resultados que le permiten al autor presentar las siguientes conclusiones:

1. EL análisis de herramientas y tecnologías existentes aportó la base tecnológica para el desarrollo de la solución propuesta.
2. La aplicación de pruebas permitió validar el correcto funcionamiento del sistema desarrollado, simulando ambientes reales en diferentes entornos de ejecución.
3. La aplicación desarrollada contribuirá a elevar la socio-adaptabilidad del país, logrando así que la distribución cubana de GNU/Linux Nova tenga una buena aceptación para los usuarios que la utilizan.

Recomendaciones

Al finalizar la investigación se recomiendan un conjunto de mejoras:

- Incluir la herramienta desarrollada en el repositorio la distribución cubana de GNU/Linux Nova.
- Modificar la herramienta apt para que actualice automáticamente los paquetes comprimidos en lugar de la versión sin comprimir.
- Definir un estándar que funcione para otros sistemas de empaquetado que no sean .deb.

Bibliografía

1. **Pierra, A.** *Nova Distribución cubana de GNU/Linux: soberanía tecnológica, seguridad, criollo.* Ciudad de la Habana. : Universidad de las Ciencias Informáticas., 2011. p. 9, 18, 19, 29..

2. **Arango, S.** *Disminución del tiempo de descarga del repositorio para la distribución GNU/Linux Nova.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2012.

3. **Isotton, A.** Debian Repository HOWTO. [En línea] [Citado el: 15 de noviembre de 2015.] Disponible en:<http://www.isotton.com/software /debian/docs/repository-howto/repository-howto.html>.

4. **Obregon, A.** Repositories - Community Ubuntu Documentation. Repositories. [En línea] [Citado el: 25 de noviembre de 2015.] Disponible en:<https://help.ubuntu.com/community/Repositories>.

5. **Gómez, S.** Manual para la gestión de software: Guía definitiva para la gestión. [En línea] [Citado el: 4 de diciembre de 2015.] Disponible en:<http://docs.fedoraproject.org/es-.

6. **Ubuntu.** Componentes de los repositorios. [En línea] [Citado el: 6 de diciembre de 2015.] Disponible en: <http://doc.ubuntu-es.org/Componentes_de_los_repositorios>..

7. **Albo, M.** *Sistema para automatizar la generación de paquetes binarios de la distribución Nova.* Ciudad de la Habana. : [En línea]. Universidad de las Ciencias Informáticas, Junio 2008. p 7-9..

8. **Sánchez, C.** *En: Diccionario de informática e Internet de Microsoft.* s.l. : Ed. Mc Graw Hill. Ediciones profesionales de Microsoft. , 2000. ISBN 48-481-2893-1. p 434..

9. **Suel, T y Memon, N.** *Algorithms for Delta Compression and Remote File Synchronization.* s.l. : CIS Department Polytechnic University Brooklyn. p. 1-2.

10. **Linux.** ¿Qué son? ¿Cuál usar? [En línea] [Citado el: 12 de diciembre de 2015.] Disponible en: http://www.linux-es.org/node/1389..

11. **Comparativa:.** compresores de archivos. [En línea] [Citado el: 14 de diciembre de 2015.] Disponible en: http://articulos.softonic.com/comparativa-compresores-de-archivos..

12. **Ubuntu.** 7-Zip. [En línea] [Citado el: 18 de diciembre de 2015.] Disponible en:<http://www.guia-ubuntu.org/index.php?title=7-zip>..

13. —. P7zip. [En línea] [Citado el: 13 de enero de 2016.] Disponible en:<http://www.guia-ubuntu.org/index.php?title=P7zip>..

14. **Pavlov, Igor.** 7-Zip. [En línea] [Citado el: 14 de enero de 2016.] Disponible en:<http://www.7-zip.com.mx/>..

15. **Mennucci, Andrea.** The debdelta suite. [En línea] [Citado el: 22 de enero de 2016.] Disponible en: <http://debdelta.debian.net/html/index.html>..

16. **Mennucci, A y Pierreyves, J.** *Debdelta. Página Manual Linux.* Agosto 2009.

17. **INTECO.** *Ingeniería del software: Metodologías y ciclos de vida.* 2009.

18. **Canos, J, Letelier, P y Penadés, M.** *Metodologías Ágiles en el Desarrollo de Software.* Valencia : Universidad Politécnica de Valencia.

19. **Kniberg, H.** *Scrum y XP desde las trincheras.* 2007.

20. **Jacobson, I, Booch, G y Rumbaugh, J.** *El proceso unificado de desarrollo de software.* Madrid : s.n., 1999.

21. **Alfonso, Y.** *Configuración de la metodología OpenUp V1.0.* La Habana : Universidad de las Ciencias Informáticas, 2012.

22. **Álvarez, J y Linares, N.** *Comparación y tendencias entre metodologías ágiles y formales. Metodología utilizada en el Centro de Informatización para la Gestión de Entidades: Serie Científica de la Universidad de las Ciencias Informáticas, vol. 4, p. 17.* 2011.

23. **Booch, G, Rumbaugh, J y Jacobson, I.** *The unified modeling language user guide. Addison Wesley.* 1998.

24. **Murillo, F.** [En línea]. [En línea] 1999. [Citado el: 2 de febrero de 2016.] Disponible en: http://www.inei.gob.pe.

25. **Sierra, M.** *Trabajando con Visual Paradigm for UML. Cantabria.* 2006.

26. **LINUX.** Reviews. [En línea] 1994-2005. [Citado el: 18 de febrero de 2016.] Disponible en: http://linuxreviews.org/beginner/bash_GNU_Bourne-Again_SHell_Reference/..

27. **DEBIAN.** [En línea]. [En línea] [Citado el: 10 de marzo de 2016.] Disponible en: https://packages.debian.org/sid/Debmirror..

28. **Integrado (IDE), Entorno de Desarrollo.** [En línea]. [En línea] 2013. [Citado el: 1 de abril de 2016.] Disponible en: https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide.

29. **Martínez, I.** Modelo Conceptual/ Modelo de dominio. [En línea] 2012. [Citado el: 13 de abril de 2016.] Disponible en: http://ldc.usb.ve/~martinez/cursos/ci3715/clase6_AJ2010.pdf.

30. **IEEE.** Recommended Practice for Software Requirements Specifications. [En línea] 2009. [Citado el: 22 de abril de 2016.] Disponible en: http://www.ieee.org/index.html?WT.mc_id=hpf_logo.

31. **Ocampo, P y Montoya, A.** *Sistema de procesamiento de información de gestión digital de historias clínicas en entorno web (SPIGDATA_HC).* 2013.

32. **RUMBAUCH, J, JACOBSON, I y BOOCH, G.** El Lenguaje Unificado de Modelado Manual de Referencia. [En línea] 2000. [Citado el: 2 de mayo de 2016.] Disponible en: http://ingenieriasoftware 2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modeladomanual-de-referencia.pdf.

33. **Lasso, I.** Qué es el código fuente. [En línea] 2008. [Citado el: 4 de mayo de 2016.] Disponible en: http://www.proye ctoautodidacta.com/comics/que-es-el-codigo-fuente.

34. **Rogers, J.** *File Naming Standards for Digital Collections.* 2014.

35. **Mañas, J.** Pruebas de Programas. [En línea] 1994. [Citado el: 8 de mayo de 2016.] Disponible en: http://www.proye ctoautodidacta.com/comics/que-es-el-codigo-fuente.

Bibliografía Consultada

- **AGARWAL, R y AMALAPURAPU, S.** *An Approximation to the Greedy Algorithm for Differential Compression of Very Large Files.* Department of Electrical Engineering and Computer Sciences. 2000.
- **ALBO, M.** *Sistema para automatizar la generación de paquetes binarios de la distribución Nova*. [en línea]. Universidad de las Ciencias Informáticas. Ciudad de la Habana Junio 2008. p. 7-9.
- **AOKI, O y ECHARRIA, W.** Debian Reference (version 1) - Fundamentos de Debian. [en línea].Disponible en la Web: <http://www.debian.org/doc/manuals/debian-reference/ch-system.es.html#s-deb-format>.
- **Sánchez, C.** En: Diccionario de informática e Internet de Microsofted. Mc Graw Hill. Ediciones profesionales de Microsoft. 2000. ISBN 48-481 -2893-1. p 138.
- **COMPRESION.**ES. Compresión de Datos - Compresión Compresores de archivos, ficheros y carpetas. [en línea. Disponible en la Web:<http://www.compresion.es/compresion-de-datos/>.
- **DEBIAN.** Proyect. deb man (5). Manual Linux.
- **ECURED.** MD5. [en línea]. Disponible en la Web:<http://www.ecured.cu/index.php/MD5>.
- **ECURED.** Compresión de datos. [en línea]. Disponible en la Web: <http://www.ecured.cu/index.php/Compresi%C3%B3n_de_datos>.
- **ECURED.** Lz77. [en línea]. Disponible en la Web:<http://www.ecured.cu/index.php/Lz77>.
- **ECURED.** SHA [en línea]. Disponible en la Web:<http://www.ecured.cu/index.php/SHA>.
- **ELLIS, S y FRIELDS, P.** Fedora Core 4: Administrando *Software* con yum. [en línea]. Disponible en la Web: http://docs.fedoraproject.org/esES/Fedora_Core/4/pdf/*Software* _Management_Guide/Fedora_Core-4-*Software* _Management_Guide-es-ES.pdf>. p. 4.
- **FEDORA.** presto. [en línea]. Disponible en la Web: <http://fedoraproject.org/wiki/Features/Presto>.
- **GNU.** Tar - GNU Project - Free *Software* Foundation (FSF). [en línea]. Disponible en la Web: <http://www.gnu.org/*software* /tar/>.
- **GÓMEZ, S.** Manual para la gestión de *software:* Guía definitiva para la gestión. Edición. [en línea]. Disponible en la Web:<http://docs.fedoraproject.org/es-ES/Fedora_15/0.1/html/*Software* _Management_Guide/ch02s02.html>.

- **GZIP.** Ubuntu Manpage: gzip, gunzip, zcat. Página Manual Linux.
- **ISOTTON, A.** Debian Repository HOWTO. [en línea].Disponible en la Web: <http://www.isotton.com/*software* /debian/docs/repository-howto/repositoryhowto.html>.
- **MENNUCCI, A y PIERREYVES, J.** *Debdelta.* Página Manual Linux. Agosto 2009.
- **MENNUCCI, A.** The debdelta suite. [en línea]. Disponible en la Web: <http://debdelta.debian.net/html/index.html>.
- **MILLER, W y MYERS, E.** *A File Comparison Program*. Department of Computer Science, Tucson. 1985.
- **OBREGON, A.** Repositories - Community Ubuntu Documentation. Repositories. [en línea]. Disponible en la Web:<https://help.ubuntu.com/community/Repositories>.
- **Sánchez, C.** En: Diccionario de informática e Internet de Microsoft., ed. Mc Graw Hill. Ediciones profesionales de Microsoft. 2000. ISBN 48-481 -2893-1. p 434.
- **PAVLOV, I.** 7-Zip. [en línea]. Consultado el 9 abril 2012a. Disponible en la Web: <http://www.7-zip.com.mx/>.
- **PIERRA, A.** *Nova Distribución cubana de GNU/Linux: soberanía tecnológica, seguridad, criollo.* Universidad de las Ciencias Informáticas. Ciudad de la Habana. 2011. p 9, 18, 19, 29.
- **ARANGO, S.** Disminución del tiempo de descarga del repositorio para la distribución GNU/Linux Nova. Universidad de las Ciencias Informáticas. Ciudad de la Habana. 2012.
- **SEWARD, J.** bzip2. Página manual Linux.
- **UBUNTU.** 7-Zip. [en línea]. Disponible en la Web:<http://www.guia-ubuntu.org/index.php?title=7-zip>.
- **UBUNTU.** Componentes de los repositorios. [en línea]. Disponible en la Web: <http://doc.ubuntu-es.org/Componentes_de_los_repositorios>.
- **UBUNTU.** P7zip [en línea]. Disponible en la Web:<http://www.guia-ubuntu.org/index.php?title=P7zip>.
- **UBUNTU.** Rar - Guía Ubuntu. [en línea]. Disponible en la Web: <http://www.guia-ubuntu.org/index.php?title=RAR>.
- **ZIP.** zip (1): package/compress files. Página Manual Linux.

Anexos

Anexo 1. Entrevista realizada a la comunidad de *software* libre e instituciones.

Preguntas:

¿Tiempo aproximado que demora la descarga del repositorio para la distribución cubana de GNU/Linux Nova?

¿Cuál es la velocidad de ancho de banda con la que cuenta la entidad actualmente?

¿De qué forma acceden a los repositorios de la distribución cubana de GNU/Linux Nova?

¿Cuál es el grado de aceptación que tiene la distribución cubana de GNU/Linux Nova?

¿Cuáles son los paquetes más usados por la comunidad?

¿Cuáles son los paquetes que se deberían incluir en el repositorio minimizado, además de los que vienen por defecto?

Anexo 2. Casos de pruebas.

Tabla 19: Caso de prueba para el Caso de Uso "Descargar código fuente del repositorio".

Sección " Descargar código fuente"			
Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Descargar código fuente.	Se descarga el código fuente correctamente.	Descarga el código fuente.	1. Se selecciona el botón "Descargar".
EC 1.2 No hay código fuente disponible.	No se descarga el código fuente del repositorio.	Muestra un mensaje informando del suceso.	1. Se selecciona el botón "Descargar". 2. Se muestra el mensaje informando el suceso.

Tabla 20: Caso de Prueba para el Caso de Uso "Descomprimir archivos en el formato tar.7z

Sección "Comprimir archivos"

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Descomprimir los archivos en tar.7z.	Se descomprimen los archivos correctamente.	Descomprime el repositorio en el formato tar.7z.	1. Descomprimir archivos.
EC 1.2 No se puede descomprimir en el formato tar.7z.	No se descomprime el repositorio.	Muestra un mensaje informando del suceso.	1. Se selecciona el botón “Comprimir”. 2. Se muestra el mensaje informando el suceso