UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Herramienta de Empaquetamiento para LibreOffice

Trabajo Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autor:

Juan Carlos Sacasas Cáceres

Tutores:

Ing. Gladys Marsi Peñalver Romero

Ing. Jorge Luis Roque Álvarez

10/06/2014

Pensamiento

"La libertad no es poder elegir entre unas pocas opciones impuestas, sino tener el control de tu propia vida. La libertad no es elegir quien será tu amo, es no tener amo."

Richard Stallman

Declaración de autoría

Declaro ser autor de la presente tesis y reconozco 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_____.

_____

Juan Carlos Sacasas Cáceres

Firma del Autor

_____

Ing. Gladys Marsi Peñalver Romero

Firma del Tutor

_____

Ing. Jorge Luis Roque Alvarez

Firma del Tutor

Agradecimientos

Quisiera agradecer a cada una de las personas que de una forma u otra han influenciado en mi formación y han estado a mi lado hasta convertirme en lo que hoy soy. A todos aquellos amigos de la infancia que no olvido, a los que vinieron después y se convirtieron en parte importante de mi vida. A mi gente del Técnico Informática que hasta hoy conservo los mejores recuerdos con los que cuento, a Yuviel, Elio, Jorgito, Pedrito, Reinier y a los demás, les agradezco su entera amistad y confianza. A la gente de esta gran Universidad que compartieron y comparten conmigo en esta universidad, así como a mi grupo y en especial a aquellos que se convirtieron casi en parte de mi familia. A mi novia Dainelys, que ha sido mi más fiel amiga y juntos hemos logrados enfrentar y vencer todas las batallas que esta vida de estudiante nos ha impuesto, a ti por estar de mi lado siempre. A todas mis amistades gracias. Gracias a mis tutores que me ayudaron a que este sueño fuera realidad, gracias a Héctor, al Mendo que me ayudaron incondicionalmente.

Gracias a toda mi familia, tanto materna como paterna, a mis tías y tíos, a mis primas y primos, por ser la mejor familia que hubiese podido desear, por acompañarme, por preocuparse, por orientarme, apoyarme, por hacer de mí el Juan Carlos que soy ahora. Gracias a mi tío Paublo que desde que comenzó mi vida como estudiante siempre me ayudado y apoyado en todo lo que me ha hecho falta. Gracias a mi abuela y abuelo del alma, por ser mí ejemplo de sacrificio, de entrega, de coraje, gracias por hacer de nuestra familia lo que somos hoy. Muchas gracias a las personas más importantes de mi vida, gracias a mis dos padres, a Juan Carlos Sacasas, al hombre que me hizo y me dio el apellido por el cual me conocen muchas personas, y a mi padre Castillo que ha sido para mí un gran ejemplo en toda mi vida, me ha cuidado, querido y me ha dado este tamaño que tengo; por otro lado mi madre, la mejor madre del mundo, la mujer más capaz, emprendedora y sacrificada que haya conocido, a ti mami te debo la vida y todo lo que soy, gracias por tu amor, sabiduría, comprensión y no acabaría de agradecer jamás todo lo que has hecho por mí, solo deseo tenerte a mi lado por siempre y poder dedicarte todo lo que consiga en mi vida.

Dedicatoria

Dedico el presente trabajo de diploma a mi abuela, mi abuelo, mi padre y mi queridísima madre, quienes han sido mi principal apoyo en la construcción de este sueño.

Juan Carlos Sacasas Cáceres

Resumen

Debido a los altos precios de licencias y productos informáticos, Cuba se encuentra en un proceso de migración hacia productos de código libre. Proceso que surge en la Universidad de las Ciencias Informáticas con la creación del Centro de Soluciones Libres. Centro que cuenta con el departamento de Servicios Integrales de Migración, Asesoría y Soporte en el cual se encuentra el proyecto de Personalización Ofimática, que tiene como objetivo desarrollar nuevas herramientas para la *suite* ofimática LibreOffice.

Según las deficiencias e inconformidades encontradas en la *suite* ofimática por parte del personal que lleva a cabo la migración en el país, se propone como solución del presente trabajo de diploma desarrollar una herramienta que permita empaquetar la personalización cubana de LibreOffice. Para ello fue necesario el estudio de las diferentes variantes y herramientas que se utilizan en el proceso de empaquetamiento. La herramienta a desarrollar es guiada por la metodología de desarrollo SXP e implementada con el lenguaje de programación Java; garantizando con ella la automatización del proceso de integración, compilación y el empaquetamiento de LibreOffice. La herramienta desarrollada garantiza la integración de los diferentes componentes realizados en el proyecto Personalización Ofimática al código fuente de LibreOffice, y así obtener la nueva personalización cubana.

Palabras claves: automatización, compilación, empaquetamiento, *suite* ofimática.

Índice de contenido

Índice de Tablas

Índice de Ilustraciones

Índice de Anexos

Introducción

El desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) han influenciado en el crecimiento de las organizaciones, corporaciones y entidades, asegurando la sustentabilidad en tiempo y espacio (1). Muchas empresas insertan la actual *suite* ofimática dominante en el mercado *Microsoft Office*, la cual posee sus propios formatos cerrados de documentos para cada uno de sus programas.

Aunque *Microsoft Office* es utilizada por más del 80% de las empresas del mundo, afronta una fuerte competencia por parte de *OpenOffice*[1], LibreOffice (LO), IBM *Lotus Symphony*[2], *Kingsoft Office*[3], *Foxit Office*[4], *Google Docs*[5] e *iWork*[6].

Debido a los costos y las grandes inversiones que representa para Cuba el pago de licencias privativas, a partir del año 2004 se estableció una política de migración a sistemas y aplicaciones libres en todo el país. Con el objetivo de hacer cumplir esta política, surge en la Universidad de las Ciencias Informáticas (UCI) el Centro de Software Libre (CESOL), en el que se encuentra el departamento de Servicios Integrales de Migración, Asesoría y Soporte (SIMAYS).

SIMAYS está especializado en brindar servicios relacionados con la migración a código abierto como asesoría, consultoría, capacitación y soporte técnico. Además desarrolla aplicaciones de apoyo al proceso de migración a código abierto.

Uno de los proyectos del departamento es Personalización Ofimática (PO), creado con el objetivo de desarrollar nuevas herramientas de la *suite* ofimática LO; que permitan garantizar el soporte en nuestro país. Además pretende alcanzar la soberanía tecnológica a través de la personalización cubana en el entorno del código abierto, para emprender desarrollos relativos a las herramientas ofimáticas para satisfacer las necesidades de las empresas e instituciones cubanas.

[1] Es una *suite* ofimática libre que incluye herramientas como procesador de textos, hoja de cálculo, presentaciones, herramientas para el dibujo vectorial y base de datos.
[2] Es una *suite* ofimática libre desarrollada por Lotus *Software* (compañía de *software* estadounidense).
[3] Es una *suite* ofimática desarrollada por el desarrollador de *software* Kingsoft en Zhuhai, China.
[4] Es una *suite* ofimática que combina *software* de dos compañías: Foxit y Kingsoft.
[5] Google Docs es un procesador de texto online que te permite crear y dar formato a documentos de texto.
[6] iWork es la suite ofimática de aplicaciones creada por Apple para el sistema operativo Mac OS X y iOS.

Durante el proceso de migración, se han encontrado deficiencias e inconformidades por parte del personal encargado de trabajar con la *suite* ofimática, dentro de las deficiencias encontradas se tiene que: las nuevas versiones de LO no traen incluido todas las macros[7], componentes[8] y herramientas creadas por el proyecto PO, como por ejemplo, el Asistente de Calendario. Todos los tipos de fuentes no están incluidos en LO, por ejemplo no tiene incluido Arial, *Times New Roman*, por lo que hay que instalarlos aparte, teniendo como consecuencia que el usuario con un nivel bajo de conocimientos informática no pueda realizar el trabajo deseado con los tipos de fuentes más conocidos. No se encuentra el idioma español por defecto por lo que no contiene la corrección ortográfica y gramatical, y el usuario no tiene dominio de cómo instalar los componentes a través de la terminal de Linux. La ayuda que contiene no ofrece al usuario todo lo necesario, pues para obtener información más detallada de la *suite* se hace una búsqueda a través de Internet en sitios como ***Welcome to LibreOffice Help![9]*,** y *fórums* de discusiones de LO referente al tema, pero el acceso a Internet es limitado para la mayoría de las empresas e instituciones cubanas. Todos estos elementos antes mencionados traen como consecuencia la resistencia al cambio a la nueva tecnología ofimática, por lo que se hace necesario integrar los componentes desarrollados en el proyecto PO a la suite ofimática LO.

Partiendo de esta problemática se plantea como **problema científico**: ¿Cómo integrarle a LibreOffice los componentes desarrollados en el proyecto Personalización Ofimática para lograr una mayor aceptación de la *suite* por parte de los usuarios en el proceso de migración a código abierto?

Objeto de estudio

Proceso de empaquetamiento en GNU/Linux.

Campo de acción

Proceso de empaquetamiento para “LibreOffice”.

Idea a defender

El desarrollo de una herramienta de empaquetamiento de los componentes desarrollados en el proyecto Personalización Ofimática permitirá lograr una mayor aceptación de la *suite* por

[7] Los macros son grupos de instrucciones que se ejecutan secuencialmente y se utilizan para economizar tareas.

[8] Uno o varios elementos de un sistema de *software* que ofrece un conjunto de servicios, o funcionalidades, a través de interfaces definidas.

[9] Sitio oficial de LibreOffice, https://help.libreoffice.org.

parte de los usuarios en el proceso de migración a tecnologías libres.

Objetivo general

Desarrollar una herramienta de empaquetamiento que permita integrar al código fuente de LibreOffice los componentes desarrollados en el proyecto Personalización Ofimática.

Objetivos específicos

- Caracterizar las principales herramientas en el proceso de empaquetamiento para LibreOffice.
- Analizar y diseñar la solución propuesta.
- Implementar la herramienta de empaquetamiento para la *suite* ofimática.
- Probar la herramienta de empaquetamiento para la *suite* ofimática.

Tareas de investigación

- Revisión bibliográfica asociada a los conceptos y herramientas del proceso de empaquetamiento en las diferentes distribuciones de GNU/Linux y en LibreOffice para conocer las principales variantes de empaquetamiento.
- Realización de una guía de compilación para LibreOffice.
- Definición de los requisitos funcionales y no funcionales para el desarrollo de la solución.
- Análisis y diseño de la herramienta para el empaquetamiento de LibreOffice.
- Implementación de las funcionalidades definidas.
- Diseño y ejecución de casos de pruebas para verificar las funcionalidades implementadas.

Métodos de Investigación

Analítico-Sintético: es utilizado con el fin de analizar un conjunto de libros, artículos y documentos sobre las variantes y herramientas para el empaquetamiento, garantizando la extracción de las ideas y elementos importantes vinculados con la investigación. Además es utilizado en el estudio de guías referentes al proceso de empaquetamiento, y en el análisis de los conceptos definidos en el marco teórico.

Histórico-Lógico: es utilizado en la consulta bibliográfica y desarrollo histórico referente a las

herramientas que permitan realizar los procesos de compilación y empaquetamiento, su evolución y comportamiento.

Este trabajo se encuentra estructurado de la siguiente forma:

Capítulo 1: Fundamentación teórica: En este capítulo se definen los principales conceptos asociados al objeto de la investigación. Se especifican las variantes y herramientas a utilizar para empaquetar. Se confecciona una guía de compilación para LibreOffice. Se realiza un estudio las herramientas, tecnologías y la metodología de desarrollo de *software* a utilizar.

Capítulo 2: Análisis y diseño de la solución: En este capítulo se describen detalladamente la propuesta de solución, se definen las características y funcionalidades de la herramienta a implementar. Además se realiza el análisis y diseño de la estructura de la herramienta, definiendo la arquitectura y patrones de diseño aplicados.

Capítulo 3: Implementación y pruebas de la solución: En este capítulo se realiza la implementación de la solución propuesta. Se define el estándar de código a utilizar en el desarrollo de la solución, y además se muestra el diseño de los casos de pruebas de aceptación efectuadas a la herramienta de empaquetamiento para garantizar su calidad. Se define el aporte que brinda la herramienta para el proyecto de PO.

Capítulo 1: Fundamentación teórica

En este capítulo se plasman los principales conceptos asociados a la investigación. Se realiza un análisis de las variantes y herramientas de empaquetamiento. Se presenta una guía de compilación para LO, confeccionada con el objetivo de indicar a los usuarios los pasos a seguir para lograr una correcta compilación del código fuente de LO. Se propone el lenguaje de programación, herramientas y metodología de desarrollo a utilizar en la propuesta de solución.

1.1 Conceptos fundamentales

La presente investigación hace referencia a los siguientes conceptos, para un mejor entendimiento de la aplicación en desarrollo.

Ofimática (automatización de oficinas)**:** Es el conjunto de técnicas, métodos y equipos utilizables en el trabajo de oficina de cualquier entidad. Es un sistema automatizado de información para la oficina, trata de realizar las tareas de la oficina tradicional por medio de sistemas de ordenadores (2).

***Suite* Ofimática o paquete ofimático:** Es el conjunto integrado de aplicaciones y utilidades de *software* que permiten crear archivos y facilitar la edición, impresión, organización e interrelación con otras aplicaciones que están dentro del mismo paquete (3).

LibreOffice: Es una *suite* ofimática libre y gratuita, compatible con *Microsoft Windows*, *Mac* y GNU/Linux. Cuenta con un procesador de texto (*Writer*), un editor de hojas de cálculo (*Calc*), un creador de presentaciones (*Impress*), un gestor de bases de datos (*Base*), un editor de gráficos vectoriales (*Draw*), y un editor de fórmulas matemáticas (*Math*) (4).

Compilador: Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo está escrito en lenguaje de máquina, aunque también puede ser un código intermedio, o simplemente texto. Este proceso de traducción se conoce como compilación (5).

Código fuente: Todos aquellos ficheros necesarios para construir el ejecutable que finalmente necesita la aplicación. Usualmente estos ficheros son de texto plano (6).

Paquete de código fuente: Contiene el código fuente de los programas necesarios para compilarlos e instalarlos. Este conjunto de ficheros fuentes suelen ser compactados, algunos

formatos son: .tar (compactado), .tar.bz2 (empaquetado con .tar y comprimido con .bzip2), .rar (comprimido), .zip (comprimido) (6).

Empaquetamiento de *software*: Es el proceso de preparar en paquetes los ficheros y es realizado por los desarrolladores del *software* antes su distribución (7).

Un **paquete .deb**, es un archivo empaquetado y comprimido que contiene toda la información necesaria para instalar un *software* mediante cualquiera de las herramientas compatibles ***dpkg*** o ***apt***. Está compuesto por tres archivos que contienen la información que guarda el paquete, estos archivos son el ***control.tar.gz***, el ***data.tar.gz*** y el ***debian-binary***. Los dos primeros son ficheros comprimidos, uno contiene información y el otro los archivos que componen el programa empaquetado mientras que el último indica el número de versión del paquete .deb construido (7).

1.2 Variantes de empaquetamiento:

Para realizar el empaquetamiento existen diferentes variantes, las cuales contienen herramientas que posibilitan realizar este proceso. La presente investigación se enmarca en las variantes libres utilizada en las distribuciones Nova y Debian, ya que mediante este proceso se puede tomar el código fuente de una aplicación, compilarlo, y generar un "paquete". Para describir las variantes de empaquetamiento es necesario conocer cómo se suministran los paquetes en GNU/Linux. Estos paquetes normalmente son suministrados en tres extensiones de formatos, los cuales son mencionados a continuación:

- RPM (*Red Hat Package Manager,* en español Herramienta de administración de paquetes): se utilizan en distribuciones basadas en *Red Hat*[10]*, Fedora*[11]*, Mandriva*[12], pueden tener tanto binarios como código fuente (7).
- Sistema de gestión de paquetes (.deb): distribuciones basadas en Debian[13]. Contiene los binarios (7).
- Archivo comprimido (.tar.gz): contienen código fuente empaquetado y comprimido, no puede ser instalado directamente sino que primero se debe compilar (8).

[10] Es la compañía responsable de la creación y mantenimiento de una distribución del sistema operativo GNU/Linux que lleva el mismo nombre: Red Hat.
[11] Es una distribución Linux para propósitos generales basada en RPM, que se caracteriza por ser un sistema estable.
[12] Es una distribución Linux publicada por la compañía francesa Mandriva destinada tanto para principiantes como para usuarios experimentados.
[13] Es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en *software* libre.

En la siguiente investigación se hace referencia a las variantes de empaquetamiento para archivos de formato .deb, ya que la herramienta a implementar tiene que cumplir con la generación de los paquetes en este formato una vez compilado el código fuente de LO.

1.2.1 Variante de empaquetamiento para Nova

En el proceso de empaquetamiento para Nova se debe cumplir como principal requerimiento el acceso a sus repositorios. Dicha variante de empaquetamiento consta de seis pasos, los cuales se muestran detalladamente a continuación:

- **Preparando el entorno de trabajo.**

Primero se instalan las herramientas que facilitarán el trabajo para el proceso de empaquetamiento. Muchas de ellas son utilizadas para empaquetar en sistemas basados en Debian, pero por sí solas no pueden realizar dicho proceso, pues por lo general el empaquetamiento se realiza con la unión de varias de ellas. A continuación se especifican dichas herramientas:

- *build-essential*: Es un metapaquete que depende de libc6-dev, gcc, g++, *make* y *dpkg-dev*. Este último contiene herramientas similares a *dpkg-buildpackage* y *dpkg-source* que se usan para crear, desempaquetar y construir paquetes binarios o fuentes.
- *devscripts*: Contiene muchos *scripts*[14] que facilitan considerablemente las tareas de un mantenedor de paquetes. Algunos de los más usados son *debdiff*, *dch*, *debuild* y *debsing*.
- *ubuntu-dev-tools*: Contiene una colección de *scripts* (al igual que *devscripts*), pero específicos para Ubuntu. Contiene herramientas como *update-maintainer*, *dgetlp*, *what-patch, pbuilder-dist*.
- *debhelper*: Son un conjunto de scripts que realizan tareas comunes de empaquetado.
- *dh-make*: Se usa para crear una plantilla para el paquete.
- *diff y patch*: Se usan para crear y aplicar parches respectivamente. Su uso está muy extendido ya que son fáciles de usar, limpios y más eficientes que modificar el código fuente bruscamente.
- *quilt*: Administra los parches.

[14] Es un lenguaje interpretado que está diseñado para ser ejecutado por medio de un intérprite.

- *gnupg*: Proyecto alternativo libre que utiliza un cliente PGP, usado para firmar digitalmente archivos (incluyendo paquetes).
- *fakeroot*: Simula ejecutar comandos con privilegios de *root*. Es útil para crear paquetes binarios para uso personal.
- *lintian*: Disecciona paquetes Debian, avisa de errores y violaciones de la normativa.
- *pbuilder*: Crea un sistema *chroot* y construye un paquete en el mismo. Es un sistema ideal para comprobar si un paquete se realiza correctamente, revisar sus dependencias y construir paquetes para probar y distribuir.
- **Debianización inicial.**

Una vez preparado el entorno de trabajo lo próximo que se debe hacer es configurar las variables de entorno *Bash shell* $DEBEMAIL y $DEBFULLNAME, que son utilizadas por varias herramientas de mantenimiento de Debian para obtener el nombre y correo electrónico como se indica a continuación (9):

```
$ cat >>~/.bashrc <<EOF
> DEBEMAIL=mbonilla@estudiantes.uci.cu
> DEBFULLNAME="Marisé Bonilla Marzá"
> export DEBEMAIL DEBFULLNAME
> EOF
```

- **Construcción del directorio de trabajo.**

Para la creación del directorio de trabajo se debe cumplir con la regla que indica el llenado de paquetes de Debian: nombre paquete-versión, por ejemplo: si se tiene un programa llamado (prueba) en la versión 0.1.32, el nombre para el directorio sería prueba-0.1.32, entonces se continúa con la terminal y se crea el directorio (9):

```
$ mkdir prueba-0.1.32
```

Luego se realiza la copia de los archivos de programa dentro del directorio. Es importante que el nombre del programa sea lo más pequeño posible.

- **Comprimir el directorio de trabajo.**

A continuación se necesitará comprimir el directorio del programa. Esto se logra escribiendo en la terminal (9):

```
$ tar -cvzf prueba-0.0.1.tar.gz prueba-0.0.1.
```

- **Respaldar el programa original y crear el directorio debian.**

Es importante hacer una copia del código fuente original, por si se necesita en el proceso o simplemente se quieren deshacer cambios (9).

Para ello, se debe asociar la terminología relacionada con el empaquetamiento. Normalmente, el código de desarrollo que procede de otra fuente superior a la original, en inglés se le llama "*upstream*". Por contra, el término "*downstream*" es el empaquetador de *software* que trabaja a partir de otro desarrollo superior.

Por ejemplo: el sistema operativo Ubuntu viene directamente de Debian, por lo que Debian es considerado *upstream* de Ubuntu, pero también se puede ver de la forma contraria que Debian es el *downstream* de Ubuntu.

El *upstream* suele tener el siguiente formato (9): nombre_paquete_version.orig.tar.gz
El ejemplo sería: prueba_0.1.32.orig.tar.gz

Para realizar todo lo descrito anteriormente se emplea dh-make:

```
$ cd Escritorio/prueba-0.1.32/
$ dh_make -e mbonilla@estudiantes.uci.cu -f ../prueba-0.1.32.tar.gz -c gpl
-s
```

Posteriormente se debe tener donde se está construyendo el paquete los siguientes directorios:

prueba-0.1.32
prueba_0.1.32.orig.tar.gz
prueba-0.1.32.tar.gz

Dentro de prueba-0.1.32 se encuentra el directorio `debian`, al ejecutar `dh_make` este crea los archivos básicos en el directorio `debian`/ y algunos archivos plantilla .ex (estos son ejemplos) que quizás puedan ser necesarios o no. Por ello, se procede a eliminar los archivos .ex:

```
$ cd Escritorio/prueba-0.1.32/debian/
$ rm *.ex *.EX
```

- **Construyendo el paquete.**

Para construir el paquete se ejecuta el comando:

```
$ debuild
```

Debuild comprobará primero si el paquete *build-depends* está instalado, luego usará dpkg-*buildpackage* para compilar, instalar y construir el o los paquetes *deb* usando las indicaciones del archivo debian/rules. Si todo está bien *debuild* intentará firmar digitalmente el o los paquetes con GPG, si da un error de que no ha encontrado ninguna llave GPG significa que se ha creado correctamente el paquete .deb, pero no se tiene una llave GPG con el mismo nombre ni el correo que se haya utilizado en el archivo debian/changelog (9).

1.2.2 Variante de empaquetamiento para Debian

Para la realización del empaquetamiento en Debian se debe cumplir como requerimiento principal la conexión a Internet. Posteriormente para la creación de los paquetes de datos .deb, se necesita distribuir paquetes compilados de programas no existentes en los repositorios oficiales o nuevas versiones de ellos. Debian y muchas de sus distribuciones derivadas incluyen todas las herramientas necesarias para la creación de paquetes .deb (10).

Para visualizar el contenido del paquete se ejecuta el siguiente comando:

```
$ ar x paquete.deb
```

Para tener todas las herramientas necesarias para crear el .deb, se instala lo siguiente:

```
$ aptitude install autoconf
$ automake dh-make debhelper
$ devscripts dpkg-dev fakeroot
$ file gcc gnupg libc6-dev
$ lintian make pbuilder perl xutils
```

El último comando instalará el programa **lintian**[15] que no es necesario para la creación del paquete pero ayudará a verificar errores.

Ahora se puede empezar a construir el paquete .deb y se hace a partir del código fuente del **editor joe**[16] en última versión 3.7. Para descargar el código fuente se puede obtener en la URL: http://joe-editor.sourceforge.net/ (11).

[15] *lintian* programa que se utiliza para verificar si los paquetes Debian que se construyen son correctos o no.

Luego se procede a los siguientes pasos:

- Crear una carpeta con el nombre del programa:

  ```
  mkdir joe
  ```

- Crear el archivo tar.gz del código fuente a la carpeta, y se descomprime con el comando:

  ```
  tar xvzf joe-3.7.tar.gz
  ```

- Entrar en el directorio del código fuente del programa y se ejecuta el comando:

  ```
  dh_make -e usuario@correo.com -f ../joe-3.7.tar.gz
  ```

  Reemplazando el "*usuario@correo.com*" por el correo del usuario y "*../*" por la ruta al paquete con el código fuente. Al finalizar, se obtiene un archivo nuevo con extensión *orig.tar.gz* que contiene el código fuente del programa original empaquetado con los estándares de Debian y el directorio Debian con los archivos necesarios (*control*, *rules*, *changelog* y *copyright*).

Control: Metadatos del paquete (dependencias).

Rules: Especifica cómo construir el paquete.

Changelog: Registro histórico del paquete de Debian.

Copyright: Información de derechos de autor del paquete.

- Se construye el paquete con el siguiente comando:

  ```
  debuild -r fakeroot
  ```

  Luego se obtiene un nuevo paquete con el nombre original del programa y su número de versión.

- La firma del paquete requiere una llave *GPG* que corresponde al correo electrónico escrito en el paso número **3**. Para crear la llave se ejecuta el siguiente comando:

  ```
  gpg --gen-key
  ```

  Si se quiere construir un paquete sin firmar, se emplea el siguiente comando:

  ```
  dpkg-buildpackage -r fakeroot
  ```

- Finalmente se comprueba que el paquete esté correctamente creado ejecutando el código:

[16] JOE es un editor con todas las funciones basada en terminales de pantalla que se distribuye bajo la Licencia Pública General de GNU (GPL).

```
lintian -i paquete.deb
```

Si no referencia algún error, es que todo se ha realizado correctamente. En caso contrario, se recomienda revisar los pasos detalladamente desde el inicio (12).

Si se quiere instalar en el equipo, se escribe en la terminal:

```
dpkg -i joe_3.7-1_i386.deb
```

Las variantes de empaquetamiento analizadas, muestran los pasos lógicos para generar los archivos .deb. Estas plantean los principales requerimientos a tener en cuenta para la realización del empaquetamiento. La especificación detallada de los pasos muestra toda la información que debe estar incluida en estos archivos .deb para que sean funcionales. Pero dichas variantes no son factibles para el proceso de empaquetamiento del código fuente de LO, pues LO contiene muchas dependencias de librerías. Las dos variantes antes analizadas solo son utilizadas para empaquetar programas con pocas dependencias, por lo que todo el proceso que definen estas variantes no es utilizado en el posterior desarrollo de la herramienta.

1.3 Herramientas para empaquetar

El empaquetado y compresión de los ficheros hacen que los mismos ocupen menos espacio en disco. Actualmente existen herramientas para el proceso de empaquetamiento, a continuación se hace referencia a tres de ellas:

***Debian Package Maker**:* es una herramienta para crear paquetes **Debian** de forma gráfica, sin necesidad de comandos en la terminal.

Incluye los campos habituales que puedes encontrar en cualquier .deb, como nombre del paquete, versión, descripción, así como una lista de las dependencias que necesitará para poder funcionar, y los conflictos que puede tener con otros paquetes. Además se pueden añadir diferentes *scripts* para automatizar procesos (13).

Se logra construir paquetes Debian usando dos modos:

- Modo Target: especifica manualmente el directorio de instalación y los archivos que se instalan en el directorio de instalación de destino.

- Modo fuente: se configura el código fuente, se compila, luego se construye un paquete usando las reglas de destino y normas que son necesarias para automatizar todo el

proceso.

Proporciona una interfaz de usuario para la información de los paquetes que se utilizará como archivo de control para crear un paquete basado en Debian (14).

También proporciona diferentes pestañas para agregar dependencias de paquetes:

- Dependencias.
- Pre-dependencias.
- Recomendados.
- Sugeridos.
- Saltos.
- Conflictos.
- Reemplaza.
- Proporciona.

Deb Creator: es un programa para compilar código fuente y para crear paquetes .deb. (15). Es muy fácil de usar, te permite personalizar el paquete con dirección de correo, nombre, versión de la aplicación e incluso una pequeña descripción (15).

En el apartado técnico, *Deb Creator* incluye campos donde indicar dependencias, licencia, arquitectura, conflictos, y otras indicaciones. Posibilitará al usuario instalar el paquete .deb generado.

***Alien* (aplicación)**: es un programa de ordenador que permite convertir entre diferentes formatos de paquetes de Linux. Soporta conversiones entre Linux Standard Base, RPM, .deb, *Stampede* (.slp), *Slackware* (.tgz).*Tarball* (tar.bz2) y tar.gz (16).

La herramienta Alien puede resultar realmente útil en el caso de instalar una aplicación que aún no haya sido empaquetada para el sistema de paquetes. Alien se encuentra en los repositorios oficiales, así que si se utiliza Ubuntu o cualquier derivado, solo se instalará

mediante la terminal o mediante el Synaptic (16).

Después del análisis de las principales características de estas herramientas, se puede concluir que la utilización de las mismas es para realizar la compilación y empaquetamiento de archivos o proyectos pequeños, donde los mismos contengan poca información. Por lo que no son factibles para lograr el empaquetamiento del código de LO, ya que el código fuente de LO contiene un gran número de dependencias y librerías para lograr un buen funcionamiento del mismo.

1.4 Guía de compilación para LibreOffice

Con el objetivo de definir los pasos de cómo realizar la compilación de LO una vez integrado una macro, componente o cualquier herramienta, se confeccionó una guía de compilación. Esta guía tiene como punto fundamental darle a conocer al usuario donde se encuentran los binarios de ejecución por subsistemas para lograr la ejecución de LO una vez compilado. La misma está orientada a cualquier distribución GNU/Linux, y brinda a cualquier usuario de manera sencilla, rápida y confiable cómo lograr el proceso de compilación del código fuente de LO.

La guía de compilación está conformada por dos variantes de compilación, una con conexión a Internet y otra sin conexión Internet. La primera variante cuenta con cuatro pasos y la otra cuenta con tres pasos.

Para más información de estas dos variantes de como compilar LO (Ver el **Anexo 1)**.

1.5 Metodología de Desarrollo de *Software*

La metodología definida para la elaboración de la solución propuesta es SXP, debido a que es la metodología que se aplica en el departamento SIMAYS, y además es apropiada para el proyecto según sus características. SXP se caracteriza por una programación rápida partiendo de iteraciones incrementales con ciclos cortos, fomentando el desarrollo de la creatividad y responsabilidad entre los miembros del equipo.

SXP es apropiada para proyectos de corta duración con requisitos cambiantes o no bien definidos, donde prevalezca la retroalimentación entre el cliente y el equipo de trabajo. El desarrollo se realiza en iteraciones cortas (*sprints*) a lo largo de tres fases, dándole cumplimiento a un grupo de actividades de las que se generan una serie de artefactos que

documentan el proceso de desarrollo, obteniendo una versión del producto con nuevas funcionalidades (17).

1.6 Lenguajes de Programación

En la herramienta a implementar se definió como lenguaje de programación *Java* por las siguientes razones:

- Es un lenguaje de sintaxis sencilla, orientada a objetos e interpretada, que permite optimizar el tiempo y el ciclo de desarrollo (compilación y ejecución).
- En *Java* el programador no tiene que preocuparse de la gestión de la memoria, pues se encarga de dar memoria necesaria a la hora de crear objetos y de liberarla cuando estos ya no se referencian en el dominio del programa (cuando ninguna variable apunta al objeto).
- En *Java* se crean instancias de clases para representar tipos de datos complejos (**¡Error! No se encuentra el origen de la referencia.**).
- Es un lenguaje multiplataforma con el cual se pueden desarrollar programas que se ejecuten sin problemas en sistemas operativos como *Windows, GNU/Linux, Mac y Unix*.
- En la actualidad incluye un gran número de librerías para múltiples trabajos como: tratamiento de excepciones, hilos para el procesamiento concurrente.
- *Java* y *Python* constituyen los únicos lenguajes en los que hasta el momento se han desarrollado asistentes (*wizards*) para LO.
- Es un *software* de distribución libre, no es necesario pagar una licencia para poder comenzar a desarrollar en este lenguaje (19).
- Los métodos nativos se ejecutan de forma más rápida usando la Máquina Virtual de *Java*.
- Interpreta el código fuente en *Bytecode*[17], por lo que *Bytecode* es ejecutado en la máquina virtual y trabaja posteriormente con la máquina física.

[17] Es un código intermedio más abstracto que el código máquina.

1.7 Herramienta de desarrollo

Como entorno de desarrollo integrado se decidió utilizar Netbeans IDE en su versión 7.4, pues posee compatibilidad con el lenguaje de programación seleccionado descrito anteriormente.

- Netbeans permite desarrollar aplicaciones de escritorio de una manera rápida y sencilla con lenguajes como Java, C/C++, XML, HTML, PHP, Groovy, Javador, JavaScript y JSP.
- Es una herramienta libre y de código abierto. Posee una gran comunidad de usuarios y desarrolladores en todo el mundo (20).
- Netbeans es fácil de instalar y de uso instantáneo; además se ejecuta en varias plataformas incluyendo Windows, Linux y Mac OS X.

1.8 Herramienta de modelado

La herramienta de Ingeniería de *Software* Asistida por Computadoras (CASE) que se utilizó es Visual Paradigm en su versión 8.0.

- Es la herramienta de modelado utilizada en la institución.
- Esta poderosa herramienta facilita la generación de código, lo que facilita el trabajo de los desarrolladores, y además está disponible en múltiples plataformas.
- Posibilita la realización de diagramas de clases, la generación de documentación y de código base para diferentes lenguajes de programación como Java, C# y PHP, además de permitir la integración con entornos de desarrollo integrado (IDE) (21).
- Se integrará con el IDE seleccionado (Netbeans) para facilitar el modelado de las clases y será utilizada en su versión para GNU/Linux.

1.9 Herramientas de desarrollo colaborativo

Las herramientas de desarrollo colaborativo permiten llevar un control de las versiones realizadas al producto. Para ello se utiliza *Subversion* como herramienta para el sistema de control de las versiones, y como servidor, *Rapidsvn.* Pues ambas son utilizadas en el departamento y en la universidad para el entorno de trabajo en la producción.

Subversion: Es un sistema de control de versiones creado en el año 2000 por la comunidad y desarrolladores de *CollabNet, Elego, VisualSVN, WANdisco*. Fue diseñado para sustituir a

CVS (*Concurrent Version System*). Se le conoce también como svn por ser el nombre de la herramienta utilizada en la línea de comandos (22).

Dentro de sus principales características se encuentran las siguientes:

- Mantiene versiones no solo de archivos, sino también de directorios.
- Es un sistema de control de versiones libre bajo una licencia de tipo Apache/BSD.
- Es un sistema general que se puede utilizar para administrar cualquier conjunto de ficheros.
- Los archivos versionados no tienen un número de revisión independiente.
- Hace un mejor uso del ancho de banda ya que en las transacciones se transmiten solo las diferencias y no los archivos completos.
- Realiza el seguimiento de las versiones de proyectos completos.

***RapidSVN*:**

- Es un cliente de interfaz gráfica para la comunicación con *Subversion*.
- Está distribuido bajo licencia GPL.
- Facilita el mantenimiento de las diferentes versiones de ficheros, desde una interfaz sencilla e intuitiva, y se encuentra disponible para plataformas *Windows, Linux, MAC OS X y Solaris*.
- Es una herramienta rápida y efectiva.

1.10 Herramienta de diseño gráfico

Como herramienta para el diseño gráfico de las interfaces se utilizó Pencil, que es un *software* gratuito y de código abierto para diseñar gráficos y animarlos en dos dimensiones.

La interfaz gráfica de Pencil dispone de las herramientas básicas para crear elementos gráficos, ya sea en formato de mapa de bits o vectorial, opciones de las herramientas tan sencillas como el tamaño, el color, una práctica paleta de colores y varias opciones de visualización (23).

1.11 Conclusiones del capítulo

En el presente capítulo se presentan los conceptos fundamentales relacionados con la propuesta de solución. Del estudio de las variantes de empaquetamiento; se concluye que ninguna resulta factible para solucionar la problemática existente, ya que solo son aplicables en áreas específicas del sistema. El análisis de las herramientas como *Debian Package Maker, Deb Creator y Alien*, aunque hacen los procesos de integración, compilación, y empaquetamiento de forma individual, sirven para analizar cómo funcionan estos procesos; pero no son factibles para el proceso de compilación de LO. Se identifican como herramientas a utilizar el NetBeans como IDE de programación en su versión 7.4, Visual Paradigm como herramienta de modelado UML en su versión 8.0, como sistema de control de versiones *RapidSVN*. Se propone a utilizar como lenguaje de programación Java, y como metodología de desarrollo SXP.

Capítulo 2: Análisis y diseño de la solución

En el presente capítulo se analizará la propuesta de solución guiada por la metodología de desarrollo a utilizar. Se definirán los requisitos funcionales y no funcionales en la Lista de Reserva de Producto (LRP) y se confeccionarán las Historias de Usuarios (HU). Se definirá la arquitectura y los patrones de diseño a utilizar. Se exponen y describen los diagramas de clases y de paquete.

2.1 Propuesta de solución

Se propone desarrollar una herramienta de empaquetamiento para la *suite* ofimática LibreOffice. Esta aplicación le permite a los usuarios del proyecto PO integrar al código fuente de LibreOffice (core) una o varias extensiones, plantillas y tipos de fuentes. La misma brinda la opción de compilar el código fuente de LO, obteniéndose la nueva personalización ofimática. Una vez compilado el código posibilita la opción de empaquetar, generando los paquetes de formato .deb para que la nueva personalización pueda ser instalada en cualquier ordenador.

2.2 Características y cualidades de la herramienta

Con el objetivo de obtener detalladamente los requerimientos que debe cumplir el sistema se realiza, la identificación, los requisitos funcionales y no funcionales, los cuales brindan a los usuarios la información según sus necesidades con el objetivo de obtener una detallada especificación de los requerimientos del sistema.

Los requisitos funcionales son condiciones o capacidades que el sistema debe cumplir. Se emplean para especificar los principales servicios que el sistema proporcione. Determinan el comportamiento de la aplicación ante entradas y salidas específicas sin alterar las funcionalidades.

Los requisitos no funcionales responden a propiedades o cualidades que el *software* debe tener para que se pueda comportar de manera atractiva, confiable y segura. Suelen estar vinculados con los requisitos funcionales y son esenciales para que el producto alcance el éxito deseado.

Para un mejor entendimiento los requerimientos planteados fueron agrupados según su prioridad en la Lista Reserva Producto (LRP). Esta lista puede crecer y actualizarse a medida

que se obtienen más conocimientos acerca del producto, con la restricción de que solo puede cambiarse entre iteraciones, para lo cual se definen los requisitos funcionales y los requisitos no funcionales.

A continuación se listan los requisitos funcionales y no funcionales de la herramienta:

Tabla 1: "Lista de Reserva de Producto (LRP)".

Asignado a	Ítem *	Descripción	Estimación	Estimado por
		Prioridad	**Alta**	
Juan Carlos Sacasas	01	Generar los binarios de ejecución.	0.2	Analista-Programador
Juan Carlos Sacasas	02	Generar los paquetes .deb.	0.2	Analista-Programador
		Prioridad	**Media**	
Juan Carlos Sacasas	03	Insertar extensiones al código fuente LibreOffice.	0.1	Analista-Programador
Juan Carlos Sacasas	04	Adicionar extensiones.	0.1	Analista-Programador
Juan Carlos Sacasas	05	Eliminar extensiones.	0.1	Analista-Programador
Juan Carlos Sacasas	06	Insertar plantillas al código fuente LibreOffice.	0.1	Analista-Programador
Juan Carlos Sacasas	07	Adicionar plantillas.	0.1	Analista-Programador
Juan Carlos Sacasas	08	Eliminar plantillas.	0.1	Analista-Programador
Juan Carlos Sacasas	09	Insertar tipografías al código fuente LibreOffice.	0.1	Analista-Programador
Juan Carlos Sacasas	10	Adicionar tipografías.	0.1	Analista-Programador
Juan Carlos Sacasas	11	Eliminar tipografías.	0.1	Analista-Programador
		Prioridad	**Baja**	
Juan Carlos Sacasas	12	Listar extensiones.	0.1	Analista-Programador
Juan Carlos Sacasas	13	Listar plantillas.	0.1	Analista-Programador
Juan Carlos	14	Listar tipografías.	0.1	Analista-

Sacasas				Programador
Juan Carlos Sacasas	15	Seleccionar código fuente de LibreOffice.	0.1	Analista-Programador
Juan Carlos Sacasas	16	Seleccionar directorio de salida.	0.1	Analista-Programador
Juan Carlos Sacasas	17	Adicionar nombre del paquete.	0.1	Analista-Programador
Juan Carlos Sacasas	18	Adicionar versión del paquete.	0.1	Analista-Programador
Juan Carlos Sacasas	19	Adicionar distribución de *software*.	0.1	Analista-Programador
Juan Carlos Sacasas	20	Seleccionar el tipo de arquitectura.	0.1	Analista-Programador
Juan Carlos Sacasas	21	Adicionar autor del paquete.	0.1	Analista-Programador
Juan Carlos Sacasas	22	Adicionar correo electrónico.	0.1	Analista-Programador
Juan Carlos Sacasas	23	Adicionar descripción del paquete.	0.1	Analista-Programador
Juan Carlos Sacasas	24	Mostrar proceso de empaquetamiento.	0.1	Analista-Programador
Juan Carlos Sacasas	25	Finalizar proceso de empaquetamiento.	0.1	Analista-Programador
Juan Carlos Sacasas	26	Mostrar información del sistema.	0.1	Analista-Programador

RNF (Requisitos No Funcionales)	
Software	
1	Tener instalada la máquina virtual de Java (jdk-7).
2	Tener instalado dpkg-buildpackage.
Restricciones de implementación.	
3	Utilizar como entorno de desarrollo integrado NetBeans (version 7.4).
4	Lenguaje de programación Java.
Usabilidad	
5	El sistema debe poseer una interfaz amigable, que posibilite a los usuarios sin experiencia una rápida adaptación.
Interfaz	
6	Interfaz sencilla e intuitiva.

2.3 Funcionalidades del sistema

Para realizar la especificación de los requisitos anteriormente definidos de utilizan las HU. Las mismas son escritas por los clientes y su construcción depende principalmente de la habilidad que tengan estos para definirlas. Son escritas en lenguaje natural, estas guían la construcción de las pruebas de aceptación y son utilizadas para estimar tiempos de desarrollo.

Quedan definidas 10 HU, y a continuación se describen las dos HU de alta prioridad, las cuales le dan respuesta al 50% de las funcionalidades de la herramienta a desarrollar. Las restantes HU se encuentran en el (**Anexo 2**).

Tabla 2: HU Compilar el código fuente de LibreOffice.

Historia de Usuario	
Número: HU_1	**Nombre Historia de Usuario:** Compilar el código fuente de LibreOffice.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Alta	**Puntos Reales:** 0.1 semana
Descripción: Permitirá copiar, integrar y compilar el código fuente de LibreOffice.	
Observaciones: Para realizar la compilación el usuario debe acceder a la opción "**Cargar**", donde debe realizar lo siguiente: ✓ Seleccionar donde se encuentra el código fuente de LibreOffice. ✓ Seleccionar el directorio de salida donde se copiará el código fuente. Pasos opcionales: ✓ Si el usuario desea realizar una nueva personalización del código fuente de LibreOffice tendrá que presionar el botón "**Personalización**", donde puede integrar Extensiones, Plantillas y Tipografías al código fuente.	

Prototipo de interfaz: (Ver Anexo 3, Ilustración Empaquetando LibreOffice.)

Tabla 3: HU Generar los paquetes .deb.

Historia de Usuario	
Número: HU_2	**Nombre Historia de Usuario:** Generar los paquetes .deb.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá empaquetar la nueva personalización ofimática.	

Observaciones:

El botón **Empaquetar** posibilitará el empaquetamiento de LibreOffice (generación de los .deb). para realizar el empaquetamiento se debe realizar lo siguiente:

- ✓ Seleccionar el código fuente y el directorio de salida, en la opción **Cargar**.
- ✓ También se tiene que llenar las opciones de **Información del paquete** las cuales son:
  - Nombre del paquete.
  - Versión.
  - Distribución.
  - Seleccionar Arquitectura.
  - Autor.
  - Correo electrónico.
  - Descripción.

Si se desea realizar una nueva personalización ofimática el usuario podrá seleccionar la opción **Configuración** donde se puede integrar extensiones, plantillas o tipografías. También se deben llenar las opciones de **Información del paquete** mencionadas anteriormente.

Prototipo de interfaz: (Ver Anexo 3, Ilustración Empaquetando LibreOffice.)

2.4 Estructura del sistema

2.4.1 Arquitectura

La arquitectura de *software* es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente, son los principios que orientan su diseño y evolución (24).

Generalmente, en el proceso de desarrollo de una aplicación suelen emplearse varios estilos arquitectónicos, esto permite dar una mejor solución mediante el empleo de las ventajas que proporcionan cada uno de ellos. En el desarrollo de la herramienta de empaquetamiento se utiliza el estilo arquitectónico N-Capas o N-*Layer*.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios capas, en caso de que se deba realizar algún cambio solo se ataca al nivel requerido sin tener que revisar entre código mezclado. Dentro de las características que posee este estilo se destacan las siguientes: la mayoría de las interacciones ocurren solo entre capas vecinas, los componentes de cada capa se comunican con los componentes de otras capas a través de interfaces bien conocidas y cada nivel agrega las responsabilidades y abstracciones del nivel inferior.

La selección de dicho estilo se fundamenta en la necesidad de dividir las capas de acuerdo con su responsabilidad. Como se muestra en la **Ilustración 1**, se definieron 2 capas en un único nivel (nivel de aplicación), la capa de Presentación y la otra la capa Lógica de Negocio.

- La capa de Presentación es la que contiene la interfaz con la que el usuario interactúa con la aplicación. En ella el usuario realiza las peticiones y recibe las respuestas por parte del sistema. Esta capa debe ser amigable y fácil de usar por el usuario.
- La capa Lógica de Negocio es donde reside las funcionalidades que se ejecutan, se reciben las peticiones de la capa de presentación y se envían las respuestas tras el proceso. En esta capa es donde se establecen todas las reglas que se deben cumplir.

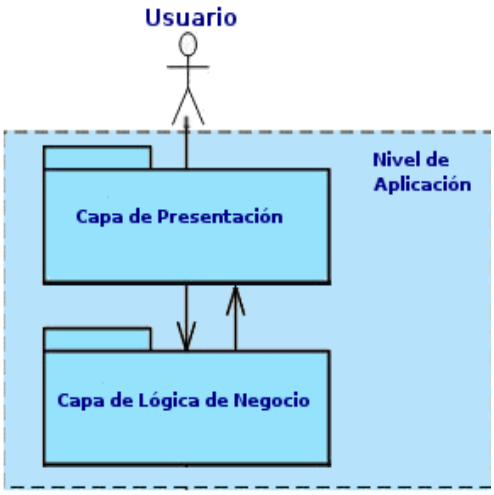


Ilustración 1: Arquitectura 2-Capas.

2.4.2 Diagrama de clases del diseño

A continuación se muestra el diagrama de clases del diseño de la herramienta a implementar. En el mismo se presentan las clases que van a contener la herramienta, así como las relaciones entre ellas. También se especifican los atributos y los métodos de cada una de las clases. Cuenta con cinco clases: Principal, Empaquetando, Controladora, Paquete y Copia.

- Principal: Contiene los métodos para seleccionar el código fuente, el directorio de salida donde se va a almacenar el nuevo código de LO, las extensiones que se quieran integrar, las plantillas y las tipografías.
- Empaquetando: Contiene los métodos que permitirán ejecutar paso a paso la compilación y empaquetamiento del código de LO.
- Controladora: Genera una carpeta debian y en ella contendrá los archivos control, changelog, compat.
- Paquete: Contiene los atributos del paquete o sea origen, destino, nombre, versión, distribución, autor, correo, descripción, y arquitectura.
- Copia: Contiene los métodos principales referentes a las diferentes copias, ya sea el código de LO, extensiones, plantillas y tipografías.

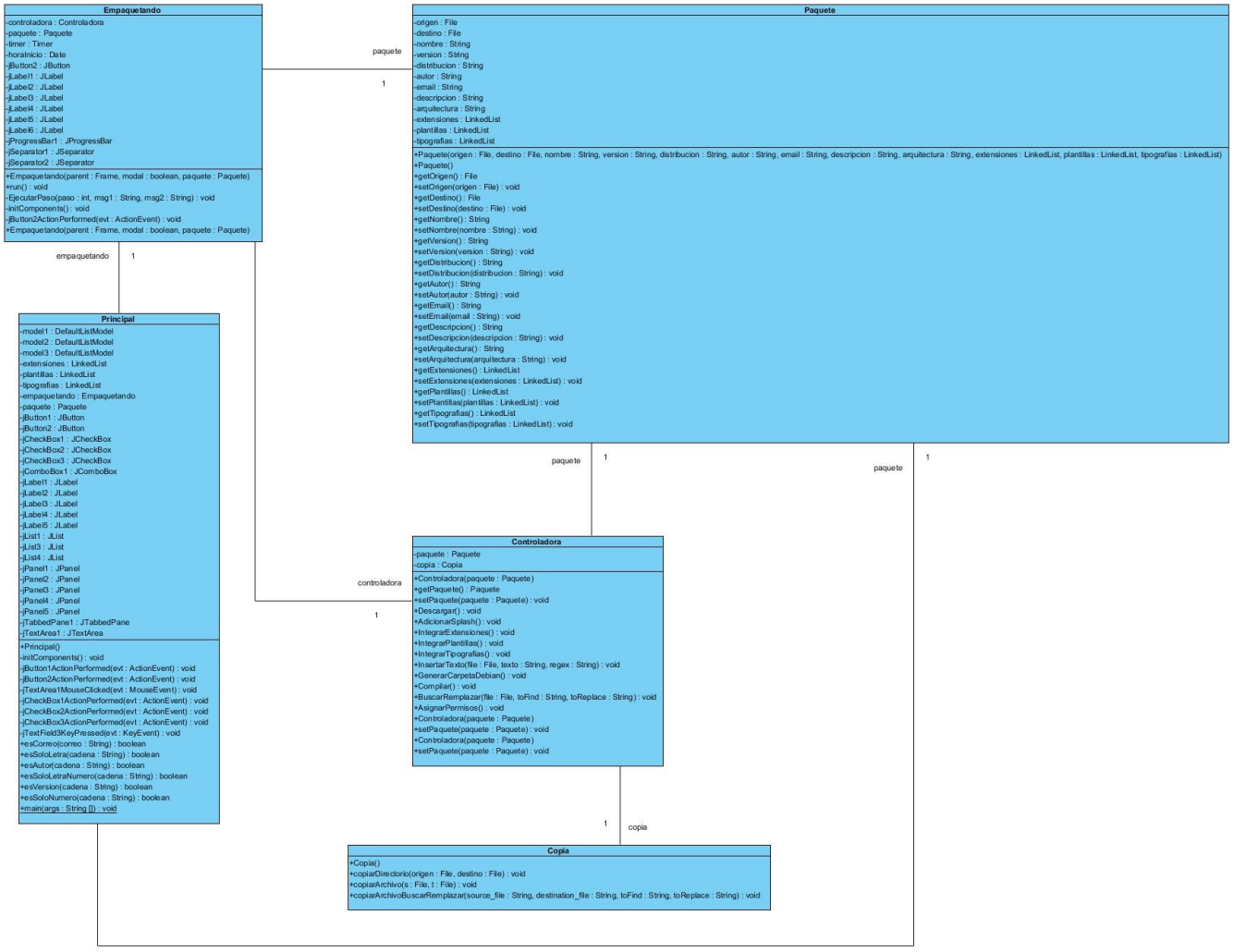


Ilustración 2: "Diagrama de clases del diseño".

2.4.3 Diagrama de paquetes

Para lograr un mayor entendimiento del funcionamiento del sistema, se hace necesario describir su estructura física. Esto se debe a que las mejoras que se le realicen posteriormente, deberán estar acopladas a la estructura que se define, proporcionando uniformidad al sistema en general. La **Ilustración 3** muestra cómo queda organizada la estructura de la herramienta según la arquitectura seleccionada (Arquitectura 2 Capas).

- Presentación contiene las clases Principal y Empaquetando.
- Lógica contiene las clases Controladora, Paquete y Copia.

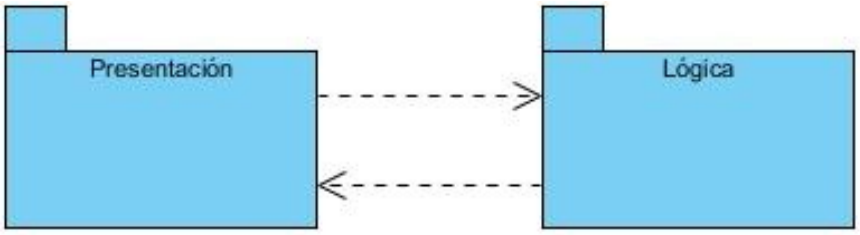


Ilustración 3: "Diagrama de paquetes del diseño".

2.4.4 Patrones de diseño

Los patrones de diseño se han convertido en una metodología por excelencia del diseño a la hora de desarrollar una solución; describen clases y objetos, así como la comunicación entre ellos en vista de resolver un problema general del diseño en un contexto específico. Por tales razones resultaron de gran ayuda durante el diseño de la herramienta los patrones siguientes:

- Experto: Este patrón se utiliza para asignarle responsabilidades a la clase que cuenta con la información necesaria para cumplir las mismas. Por citar un ejemplo, se tiene la clase Empaquetando a la que se le asignó la responsabilidad de implementar todas las funcionalidades que harán las llamadas al sistema, correspondientes al proceso de empaquetamiento.
- Creador: Este patrón se utiliza para asignarle la responsabilidad a una clase de crear instancias de otra, brindando así mejores oportunidades de reutilización, ejemplo de esto se tiene la clase Controladora a la que se le asignó la responsabilidad de crear instancias de la clase Copia para ejecutar las acciones principales.

2.5 Conclusiones del capítulo

El desarrollo de este capítulo permitió obtener los detalles de la herramienta y la definición de los requisitos tanto funcionales como no funcionales que debe cumplir el sistema. Mediante la metodología utilizada se realizó la especificación de 26 requisitos funcionales y 6 requisitos no funcionales, y las correspondientes HU. Se definió a utilizar una arquitectura n-capas con un estilo de dos capas, ya que el sistema cuenta con las capas de lógica del negocio y la de presentación. Con la realización del diagrama de clases se definieron las relaciones y métodos que fueron utilizados en la solución propuesta.

Capítulo 3: Implementación y pruebas de la solución

3.1 Introducción

Para lograr el desarrollo satisfactorio de la solución propuesta, se da en el presente capítulo la descripción del plan de iteraciones, las tareas de ingeniería y el estándar de código a utilizar en la implementación de la misma. Se presentan los casos de Prueba de Aceptación recogidos como parte de la metodología de desarrollo de *software* seleccionado, los cuales se enfocan en la verificación del cumplimiento de las funcionalidades declaradas en el proceso de concepción de la aplicación

3.2 Planificación de las Iteraciones

Una vez realizadas las HU y estimado el esfuerzo por los desarrolladores para la realización de las mismas, se hace necesario realizar la planificación de las etapas de implementación de la herramienta. Este plan contiene las HU por iteraciones, definiendo cuáles de ellas serán desarrolladas en cada iteración del proceso de implementación. La implementación de la herramienta se decide realizar en tres iteraciones, las cuales se describen a continuación:

- Iteración 1.

Esta primera iteración tiene como objetivo dar cumplimiento a las HU de prioridad alta (01, 02); que constituyen el eje central de la estructura básica de la herramienta.

- Iteración 2.

Esta segunda iteración tiene como objetivo dar cumplimiento a las HU de prioridad media (03, 04). Estas le brindarán al sistema un alto nivel de aceptación por parte del usuario ya que le permite a este integrar extensiones, plantillas y tipografías a la nueva personalización ofimática.

- Iteración 3.

La tercera y última iteración tiene como objetivo dar cumplimiento a las HU de prioridad baja (05, 06, 07, 08, 09, 10), con la implementación de estas se culmina con las funcionalidades de la herramienta propuesta. Una vez terminada esta iteración se tendrá la aplicación lista para ser utilizada.

Tabla 4: Plan de iteraciones.

Iteración	Descripción de la iteración.	Orden de la HU a implementar.	Duración total de la iteración.
1	Desarrollo de las Historias de Usuario de prioridad *Alta.*	HU_01, HU_02	8 semanas
2	Desarrollo de las Historias de Usuario de prioridad *Media.*	HU_03, HU_04	6 semanas
3	Desarrollo de las Historias de Usuario de prioridad *Baja*.	HU_05, HU_06, HU_07, HU_08, HU_09, HU_10	2 semanas

3.3 Tareas de Ingeniería (TI)

Las TI se llevan a cabo con el fin de detallar de forma más amplia las HU, facilitando con ello el entendimiento en el proceso de implementación. Cada HU puede contener una o más tareas en caso de necesitarla, explicando paso a paso las acciones que se realizan en la misma. Los puntos de estimación asignados a cada tarea son medidos por días según las características y la complejidad que posea la misma. Estas tareas definen cada una de las actividades asociadas a las HU y permiten organizar el proceso de implementación, así como conocer el grado de complejidad de cada HU. Teniendo en cuenta la cantidad de tareas asociadas (25), a continuación se exponen las TI identificadas en el desarrollo de la herramienta. La descripción de cada TI se encuentra en el (**Anexo 3**).

Tabla 5: Tareas de ingeniería.

Tareas de ingeniería	HU asociadas
Copiar el código fuente de LibreOffice.	HU-01
Compilar para generar los binarios de ejecución.	
Utilizar la clase File.	HU-02
Implementar el método Empaquetar para generar los paquetes .deb.	
Utilizar la clase JFileChooser.	HU-03
Utilizar la clase JOptionPane.	HU-04

Diseñar la interfaz seleccionar código.	HU-05
Diseñar la interfaz seleccionar directorio.	HU-06
Utilizar la clase Matcher.	HU-07
Utilizar la clase Pattern.	
Utilizar la clase ProcessBuilder.	HU-08
Utilizar la clase Timer.	
Utilizar la clase Calendar.	
Utilizar la clase Date.	
Utilizar la clase GraphicsEnvironment.	
Utilizar la clase IOException.	
Utilizar el método actionPerformed().	HU-09
Implementar el método EjecutarPaso().	HU-10

3.4 Estándar de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Los estándares de codificación deben ser implementados de forma legible y entendible, tendiendo siempre a lo práctico para los programadores que componen el equipo de desarrollo. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de *software*. La realización de buenas prácticas de programación y técnicas sólidas de codificación de alta calidad, son de gran importancia para obtener un *software* de alto rendimiento (26). A continuación se le presentará el estándar de codificación escogido para la realización de la implementación de esta investigación.

Para la capitalización de los identificadores se utilizaron los convenios:

- **Pascal**: La primera letra en el identificador y la primera letra de cada subsiguiente palabra concatenada se capitalizan, por ejemplo: IntegrarExtensiones.

- **Camel**: La primera letra en el identificador se pone en minúscula y la primera letra de cada subsiguiente palabra concatenada en mayúscula, por ejemplo: getArquitectura.

La convención Pascal se empleó en los identificadores de las clases, métodos y nombres de ficheros, mientras que la convención *Camel* es el estilo de los identificadores de las variables, atributos y parámetros. En los identificadores de las variables se tuvo en cuenta emplear su significado y obviar las abreviaturas, específicamente en las variables *booleanas* sus identificadores usa el prefijo “es”, por ejemplo: esSoloLetra; los identificadores de los métodos están en correspondencia a lo que hacen. (**Ver Anexo 4**)

Luego de implementada la solución propuesta, se procede a describir un conjunto de Pruebas de Aceptación asociadas a las HU, con el objetivo de verificar que la aplicación desarrollada cumpla con el funcionamiento esperado.

3.5 Pruebas

En el proceso de desarrollo de un *software*, la realización de pruebas constituye una actividad importante para llevar a cabo el control de la calidad del mismo.

A la herramienta desarrollada se le realizaron pruebas para comprobar su factibilidad, estas pruebas verifican el *software* en busca de errores en el manejo de las restricciones de integridad de las clases del sistema, teniendo como objetivo la verificación de discrepancias entre los requerimientos y los resultados de la ejecución del *software*. Las pruebas determinan el nivel de calidad y comprueban el grado de cumplimiento con respecto a las especificaciones iniciales del sistema (27).

En la fase de desarrollo de la metodología SXP, específicamente en el flujo de trabajo prueba se genera un artefacto denominado Caso de Prueba de Aceptación, utilizando como tipo de prueba funcional, ya que la misma fija su atención en la validación de las funciones y métodos que posee la herramienta. Se utiliza como métodos de pruebas el de caja blanca para realizar las pruebas sobre las funciones internas de la aplicación, y el de caja negra, para efectuar las pruebas sobre la interfaz de la herramienta. A continuación se muestran los Casos de Prueba de Aceptación correspondientes a las dos HU de alta prioridad implementada. Los restantes casos de pruebas se encuentran en el (**Anexo 5**).

Tabla 6: Prueba de aceptación para generar los binarios de ejecución.

Caso de Prueba de Aceptación	
Código Caso de Prueba: 01	**Nombre Historia de Usuario:** Generar los binarios de ejecución.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Probar que la herramienta compila.	
Condiciones de Ejecución: El código a compilar tiene que estar en la computadora origen.	
Entrada / Pasos de ejecución: 1. Abrir la herramienta (SELO). 2. Seleccionar código fuente del paquete. 3. Seleccionar directorio de salida. Pasos opcionales. - Integrar extensiones. - Integrar plantillas. - Integrar tipografías. 4. Insertar nombre del paquete. 5. Insertar versión. 6. Insertar distribución. 7. Seleccionar la arquitectura. 8. Insertar Autor. 9. Insertar correo electrónico. 10. Insertar descripción. 11. Se presiona el botón Empaquetar.	
Resultado Esperado: Generación de los binarios de ejecución.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 7: Prueba de aceptación para generar los paquetes .deb.

Caso de Prueba de Aceptación	
Código Caso de Prueba: 02	**Nombre Historia de Usuario:** Generar los paquetes .deb.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Probar que la herramienta empaqueta.	
Condiciones de Ejecución: Tener instalado dpkg-buildpackage.	
Entrada / Pasos de ejecución: 1. Abrir la herramienta (SELO). 2. Seleccionar código fuente del paquete. 3. Seleccionar directorio de salida. 4. Pasos opcionales. • Integrar extensiones. • Integrar plantillas. • Integrar tipografías. 5. Insertar nombre del paquete. 6. Insertar versión. 7. Insertar distribución. 8. Seleccionar la arquitectura. 9. Insertar Autor. 10. Insertar correo electrónico. 11. Insertar descripción. 12. Se presiona el botón Empaquetar.	
Resultado Esperado: Debe aparecer una ventana indicando cada uno de los pasos de la compilación y finalmente la generación de los paquetes .deb.	
Evaluación de la Prueba: Satisfactoria.	

3.6 Resultados de las pruebas de aceptación

Fueron aplicados 10 casos de pruebas de aceptación, donde se arrojaron varias no conformidades, esto se les mostrará en la Tabla 8 con una representación gráfica de la cantidad de no conformidades por iteraciones.

En la primera iteración se le aplicaron las pruebas de aceptación a 2 HU, las cuales arrojaron 10 no conformidades dándole respuestas a las mismas, en la segunda iteración se le aplicaron las pruebas a 2 HU y estas suministraron 6 no conformidades las cuales quedaron resueltas. La tercera iteración quedaron resueltas todas las no conformidades.

Tabla 8: No conformidades de los casos de aceptación por iteración.

Iteraciones	HU	No conformidades detectadas	No conformidades resueltas
Iteración 1	2	10	10
Iteración 2	2	6	6
Iteración 3	6	0	0

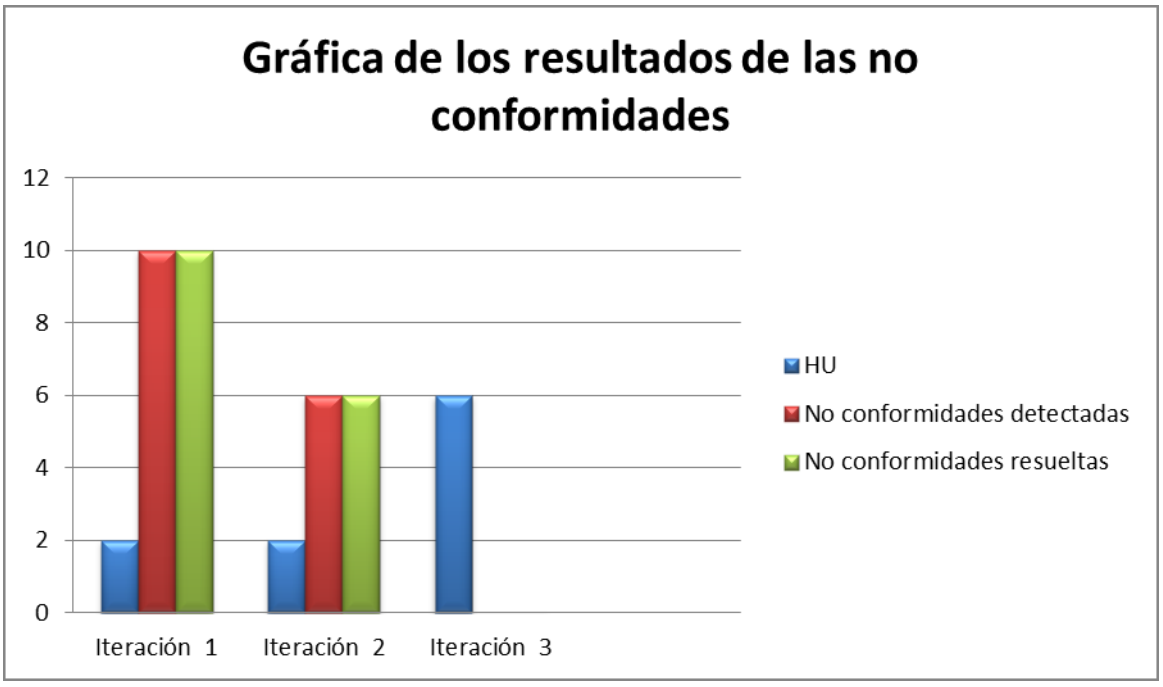


Ilustración 4: Representación gráfica de los resultados de las pruebas de aceptación.

3.7 Impacto e importancia

La herramienta desarrollada posibilita de forma fácil y segura el proceso de compilación y empaquetamiento del código fuente de LO. El personal con conocimientos de compilación y empaquetamiento puede usar esta herramienta ya que facilita de forma automatizada la integración de plantillas, extensiones y tipografías creadas en el proyecto PO. La importancia de tener una personalización ofimática radica en que todos los usuarios no tienen las mimas necesidades, en cuanto a la suite ofimática utilizada, y por ende se necesita realizar una personalización distinta para cada una de las empresas e instituciones.

3.8 Conclusiones del capítulo

El desarrollo de este capítulo permitió obtener el plan de iteraciones obteniendo de ellas una planificación de la implementación del sistema en el tiempo especificado para cada una de las iteraciones. Mediante el estilo de codificación determinado se concibe un código organizado y fluido logrando la implementación de las funcionalidades descritas en las historias de usuario. Mediante las pruebas realizadas se identificaron los errores del sistema lo que contribuyó a dar solución a los mismos para satisfacer el comportamiento esperado por el usuario final de las funcionalidades del *software* implementado.

Conclusiones generales

En la presente investigación se arribaron a las siguientes conclusiones:

- Tras el estudio de los conceptos y las diferentes herramientas que se utilizaron para dar solución al problema planteado, se pudo determinar los principales pasos y funcionalidades para el proceso de empaquetamiento de la personalización cubana de LibreOffice.
- La guía de compilación de LO permitió conocer los pasos para realizar la compilación del código fuente de LO, y saber dónde se encuentran ubicados los binarios de ejecución para la posterior utilización del código.
- La identificación de las herramientas, lenguajes y tecnologías posibilitó la creación de la herramienta que permite la compilación y empaquetamiento de LibreOffice.
- La realización del análisis y el diseño durante dichos procesos, permitió la implementación exitosa de la solución propuesta.
- La herramienta desarrollada constituye una solución para el proyecto Personalización Ofimática, que podrá integrar al código fuente de LibreOffice las extensiones, plantillas y tipografías desarrollas en el proyecto.
- Las pruebas realizadas permitieron comprobar la calidad de la herramienta, arrojando resultados satisfactorios.

Recomendaciones

Los objetivos generales de este trabajo han sido logrados, pero a lo largo de su desarrollo han surgido nuevas ideas que podrían implementarse en un futuro, de forma tal que se logre una aplicación más completa y potente. Para lo cual se recomienda:

- Implementar a esta herramienta la funcionalidad de integración de macros.
- Integrar a la herramienta una nueva funcionalidad que permita debianizar el código fuente de LO.

Referencias bibliográficas

1. *Sinergia natural en la globalización: Suite ofimática y organizaciones flexibles e inteligentes.* **Alix Aguirre Andrade, Nelly Manasía Fernández.** 3, Venezuela : Marcaibo, 2009, Vol. 15. 1315-9518.

2. Corrales, Juan Desongles. *Ayudante técnico de informática de la Junta de andalucía: Temario.* s.l.: MAD-Eduforma, 2005. Vol. II. 8466520139.

3. Purificación Aguilera López, E. A. (2011). Aplicaciones ofimáticas. Editex.

4. Características » LibreOffice. In: [online]. [Accedido 11 enero 2014]. Disponible en: http://es.libreoffice.org/caracteristicas/.

5. Compiladores, Principios, Técnicas y Herramientas - Descargar PDF. In: [online]. [Accedido 11 noviembre 2013]. Disponible en: http://www.cosaslibres.com/libro/compiladores-principios-tecnicas-y-herramientas_4586.html.

6. Camazón, J. N. (2011). *Sistemas operativos monopuesto.* Editex.

7. Montoro, Arturo Fernández. *Cámbiate a LINUX.* s.l.: RC Libros, 2011. 8493831255.

8. Christopher Negus, F. C. (2011). *Ubuntu Linux Toolbox: 1000+ Commands for Ubuntu and Debian Power Users.* (J. W. Sons, Ed.)

9. NOVA » ¿Cómo empaquetar para Nova? In: [online]. [Accedido 24 octubre 2013]. Disponible en: http://nova.uci.cu.

10. Crea tus propios paquetes .deb. - Taringa! In: [online]. [Accedido 24 octubre 2013]. Disponible en: http://www.taringa.net/posts/linux/10540999/Crea-tus-propios-paquetes-deb.html

11. Joe's Own Editor. In: [online]. [Accedido 11 febrero 2014]. Disponible en: http://joe-editor.sourceforge.net/

12. Guía de creación de paquetes Debian - packaging-tutorial.es.pdf. In: [online]. [Accedido 20 febrero 2014]. Disponible en: http://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.es.pdf.

13. Debian Package Maker: Genera paquetes DEB desde un entorno gráfico. In: [online]. [Accedido 13 febrero 2014]. Disponible en: http://linuxzone.es/2008/02/08/debian-package-maker-genera-paquetes-deb-desde-un-entorno-grafico/.

14. Debianpackagemaker – *A simple and straight forward ubuntu package maker* | Ubuntu Geek. In: [online]. [Accedido 11 febrero 2014]. Disponible en: http://www.ubuntugeek.com/debianpackagemaker-a-simple-and-straight-forward-debian-package-maker.html.

15. *Software* gratis para una sociedad libre: deb creator. In: [online]. [Accedido 11 febrero 2014]. Disponible en: http://amistosamentelinux.blogspot.com/2011/04/deb-creator.html.

16. Aprendamos Linux: Instalación y Uso de la Aplicación ALIEN en UBUNTU. In: [online]. [Accedido 11 noviembre 2013]. Disponible en: http://aprendamosconlinux.blogspot.com/2011/09/instalacion-y-uso-de-la-aplicacion.html.

17. PEÑALVER, Gladys Marsi ROMERO. SXP, metodología ágil para proyectos de *Software* Libre. Habana: Universidad de las Ciencias Informáticas, 2008.

18. Groussard, Thierry**.** *JAVA 7: Los fundamentos del lenguaje Java.* s.l. : Ediciones ENI, 2012. pág. 13. 2746073188.

19. Lenguaje de Programación en Java | Académica. In: [online]. [Accedido 13 febrero 2014]. Disponible en: http://www.academica.mx/blogs/lenguaje-programaci%C3%B3n-en-java.

20. “Netbeans IDE - *Overview*.” [Online]. [Accedido: 20 noviembre 2013]. Disponible en: http://netbeans.org/features/index.html.

21. (Paradigma Visual para UML (ME)) por Visual Paradigm International Ltd. [1 Abril 2013]. Disponible en la Web: <http://www.freedownloadmanager.org/es/>.

22. Apache Subversion. In: [online]. [Accedido 13 noviembre 2013]. Disponible en:

http://subversion.apache.org/.

23. Herramienta Pencil. [online]. [Accedido 25 febrero 2014]. Disponible en: http://www.edukanda.es/mediatecaweb/data/zip/1333/page_19.htm.

24. CARLOS BILLY REYNOSO. Introducción a la Arquitectura de *Software* [online]. Buenos Aires, 2004. [Accedido 30 marzo 2014]. Disponible en: http://carlosreynoso.com.ar/archivos/arquitectura/Introduccion.PDF.

25. PEÑALVER ROMERO, GLADYS MARSI. Propuesta de un expediente, para los proyectos productivos del Polo de *Software* Libre, de la Facultad 10. S.l.: s.n. Página 12.

26. Revisiones de código y estándares de codificación. [online]. [Accedido 25 abril 2014]. Disponible en: http://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx.

27. Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico.* Quinta Edición. s.l.: Adaptado por Darrel Ince. Pág. 282-283 Cap17.

Bibliografía consultada

1. Christopher Negus, F. C. (2011). *Ubuntu Linux Toolbox: 1000+ Commands for Ubuntu and Debian Power Users.* (J. W. Sons, Ed.)

2. Vicente Martínez García, V. M. (2005). *Modelización matemática de la sedimentación en la costa.* Universidad Jaume I.

3. Pressman, R. S. (1997). *Ingeniería del software: un enfoque práctico.* Mikel Angoar.

4. Montoro, A. F. (2011). *Cámbiate a LINUX.* RC Libros.

5. Camazón, J. N. (2011). *Sistemas operativos monopuesto.* Editex.

6. Purificación Aguilera López, E. A. (2011). *Aplicaciones ofimáticas.* Editex.

7. Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico.* Quinta Edición. s.l.: Adaptado por Darrel Ince. Pág. 282-283 Cap17.

8. Groussard, Thierry**.** *JAVA 7: Los fundamentos del lenguaje Java.* s.l. : Ediciones ENI, 2012. pág. 13. 2746073188.

9. *Sinergia natural en la globalización: Suite ofimática y organizaciones flexibles e inteligentes.* **Alix Aguirre Andrade, Nelly Manasía Fernández.** 3, Venezuela : Marcaibo, 2009, Vol. 15. 1315-9518.

10. **García, Gerardo Aburruzaga.** *Make. Un programa para controlar la recompilación.* Universidad de Cádiz : España License de Creative Common, 2008.

Anexos

Anexo 1: Guía de compilación para LibreOffice.

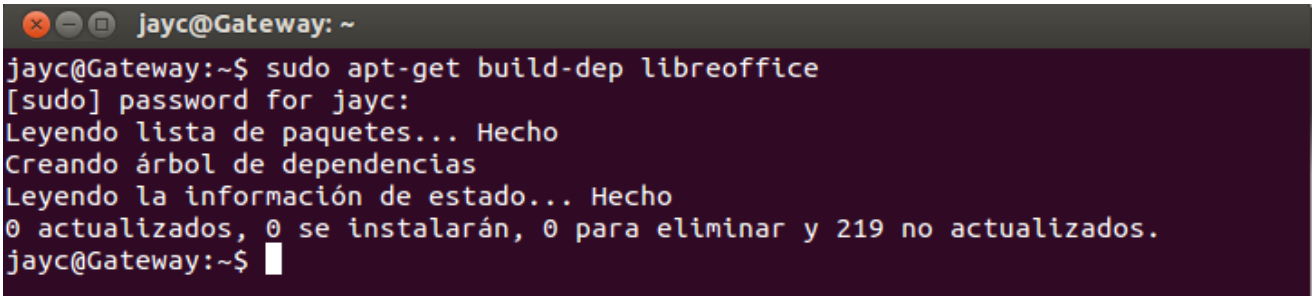
Existen dos forma de compilar LibreOffice, una teniendo conexión a internet y otra teniendo descargado en tu ordenador el código de LibreOffice (core), en la presente guía se explicará, ilustrará y mostrará cómo realizar cada uno de estos pasos para realizar una buena compilación de la *suite* ofimática LibreOffice.

Compilación de LibreOffice sin conexión a internet.

El usuario necesitará tener descargado en su ordenador el código fuente de LibreOffice (core), preferiblemente tener instalado Ubuntu 12.04, ya que le permitirá al usuario tener incluido en el repositorio todas las *sources list* (src) necesarias, la forma más factible de construir LibreOffice está en Linux. En primer lugar usted necesitará instalar las dependencias de compilación:

Para Debian / Ubuntu:

```
sudo apt- get build - dep libreoffice
```



Una vez instalado las dependencias tiene que posicionarse dentro de la carpeta **core**, en este caso la carpeta **core** se nombró como **libreoffice-4.3** y se encuentra en **home**, en una carpeta llamada **finalgit/libreoffice-4.3$**

/home/ Directorios de datos de los usuarios.

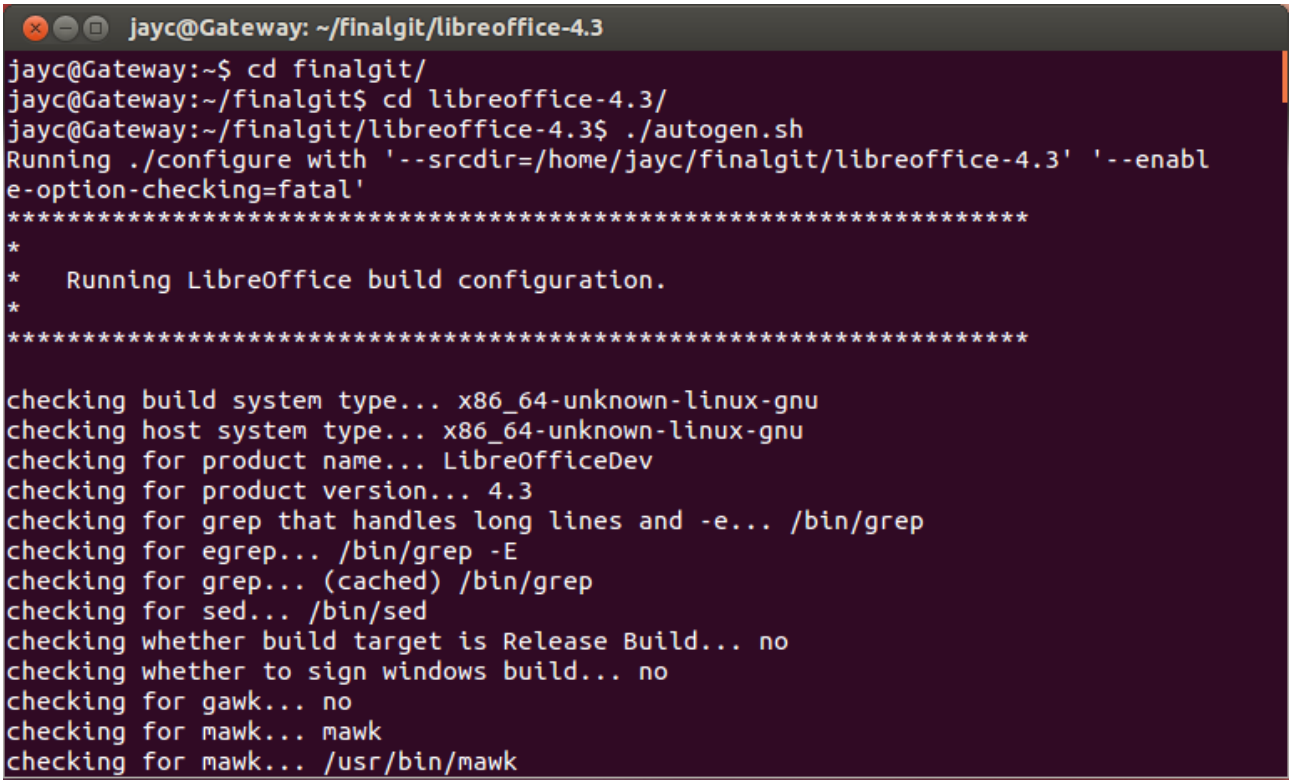
Para posicionarse dentro de la carpeta **libreoffice-4.3**, se pone en la terminal:

```
cd finalgit/libreoffice-4.3
```

cd: Comando que se utiliza para desplazarse entre las carpetas y directorios.

Realizar autogen.sh.

Cuando el usuario esté posicionado dentro de la carpeta **core** en este caso llamada **libreoffice-4.3**, tiene que ejecutar el comando **autogen.sh** que permitirá correr la configuración, chequear y configurar el código LibreOffice.



Después que haya concluido de chequear y configurar, aparecerán estas 4 opciones.

Para construir LibreOffice, ejecuta:

/usr/bin/**make**

Para obtener más información, ejecute:

/usr/ bin/**make help**

Después que la construcción haya terminado, podrá ejecutar LibreOffice:

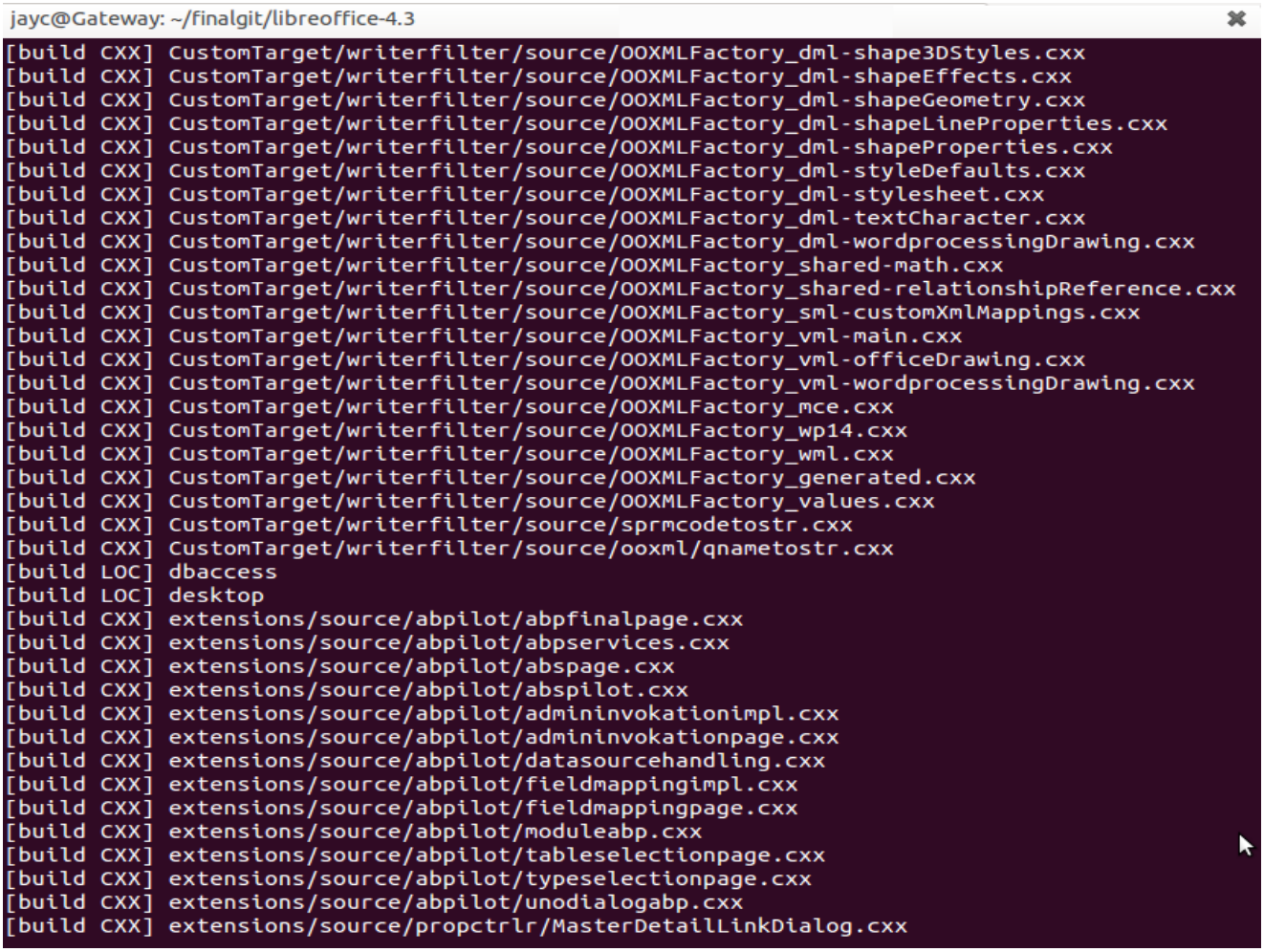
instdir/*program*/**soffice**

Si desea ejecutar la *SmokeTest*, ejecuta:

/usr/bin/**make check**

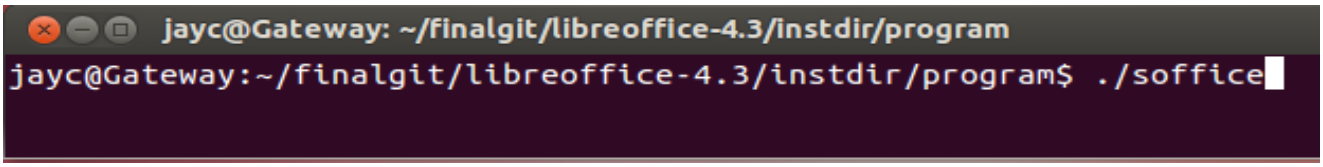
Make: Es una herramienta de gestión de dependencias que existen entre los archivos que componen el código fuente de un programa, para dirigir su re-compilación o "generación" automáticamente.

Si el usuario utiliza *make* para construir LibreOffice, después de compilado solo podrá ser usado en el ordenador donde se compiló, para que pueda ser utilizado en otros ordenadores después de compilado, se tiene que compilar con el siguiente comando, ***make fetch***. Cuando introduzcas el comando ***make fetch*** comienza la compilación y creación de todos los binarios.



La compilación puede durar de 3 a 5 horas según las propiedades del ordenador con el que se realice, una vez que haya concluido la compilación podrás ejecutar LibreOffice con el comando:

finalgit/libreoffice-4.3/instdir/program$ ./soffice



La carpeta ***instdir*** se crea cuando concluye la compilación y dentro de esta carpeta se encuentra otra carpeta llamada ***program*** que dentro contiene todos los binarios de ejecución.

Ubicación de los Binarios:

Los binarios se encuentran dentro de la carpeta ***program***.

En este caso:

/home/jayc/finalgit/libreoffice-4.3/instdir/program/soffice

Para probar que ya se tiene el código compilado se pone en terminal

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice.bin

Ubicación del binario soffice por subsistemas (*Writer*, *Calc*, *Impress*, *Math*, *Base*, *Draw*).

Writer

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice - -writer

Calc

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice - -calc

Impress

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice - -impress

Math

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice - - math

Base

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice - - base

Draw

/home/jayc/finalgit/libreoffice-4.3/instdir/program# ./soffice - - draw

Compilación de LibreOffice con conexión a internet.

El usuario necesitará conocer el espacio necesario para el almacenamiento del código fuente de LibreOffice y conocer el uso del repositorio git llamado "núcleo".

Espacio en el disco duro.

Como mínimo con todos los repositorios *git*, y la acumulación de producto y paquetes, el usuario necesitará de 7 a 10 GB de espacio en el disco, dependiendo de la plataforma, las opciones de compilación y las opciones *autogen* específicas de uso, se necesitará unos pocos GB más si se realiza una versión de depuración.

Se mostrará cómo trabajar con el repositorio ~/git. Por supuesto, el usuario es libre de elegir el directorio que desea siempre como punto de partida. Ahora se procede a cargar el "núcleo" del repositorio git:

Ejemplo:

```
$ mkdir git

$ cd git

$ git clone git://gerrit.libreoffice.org/core libo

Cloning into libo...

Remote: Counting objects: 76845, done.

Remote: Compressing objects: 100% (17328/17328), done.

Remote: Total 76845 (delta 60786), reused 74045 (delta 58579)

Receiving objects: 100% (76845/76845), 15.82 MiB | 1.17 MiB/s, done.

Resolving deltas: 100% (60786/60786), done.

$ cd libo
```

Una vez concluida la descarga del código fuente el usuario podrá continuar con los mismos pasos que fueron explicados en la **Compilación de LibreOffice sin conexión a internet**.

Anexo 2: Historias de Usuario.

Historia de Usuario	
Número: HU_1	**Nombre Historia de Usuario:** Compilar el código fuente de LibreOffice.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Alta	**Puntos Reales:** 0.1 semana
Descripción: Permitirá copiar, integrar y compilar el código fuente de LibreOffice.	

Observaciones:

Para realizar la compilación el usuario debe acceder a la opción "**Cargar**", donde debe realizar lo siguiente:

- ✓ Seleccionar donde se encuentra el código fuente de LibreOffice.
- ✓ Seleccionar el directorio de salida donde se copiará el código fuente.

Pasos opcionales:

- ✓ Si el usuario desea realizar una nueva personalización del código fuente de LibreOffice tendrá que presionar el botón "**Personalización**", donde puede integrar extensiones, plantillas y tipografías al código fuente.

Prototipo de interfaz:

Historia de Usuario	
Número: HU_2	**Nombre Historia de Usuario:** Generar los paquetes .deb.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá empaquetar la nueva personalización ofimática.	

Observaciones:

El botón **Empaquetar** posibilitará el empaquetamiento de LibreOffice (generación de los .deb). para realizar el empaquetamiento se debe realizar lo siguiente:

- ✓ Seleccionar el código fuente y el directorio de salida, en la opción **Cargar**.
- ✓ También se tiene que llenar las opciones de **Información del paquete** las cuales son:
  - Nombre del paquete.
  - Versión.
  - Distribución.

- Seleccionar Arquitectura.
- Autor.
- Correo electrónico.
- Descripción.

Si se desea realizar una nueva personalización ofimática el usuario podrá seleccionar la opción **Configuración** donde se puede integrar extensiones, plantillas o tipografías. También se deben llenar las opciones de **Información del paquete** mencionadas anteriormente.

Prototipo de interfaz:

Historia de Usuario	
Número: HU_3	**Nombre Historia de Usuario:** Integrar, adicionar y eliminar extensiones al código fuente LibreOffice.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Alta	**Puntos Reales:** 0.1 semana
Descripción: Permitirá adicionar y eliminar al código fuente de LibreOffice nuevas extensiones.	

Observaciones:

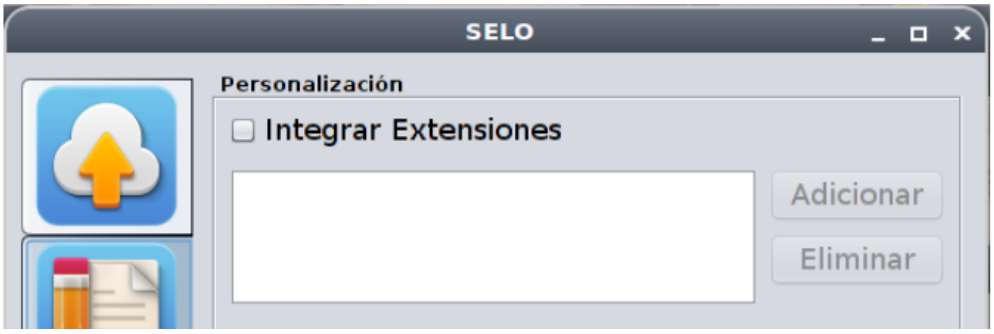
Para integrar una nueva extensión el usuario tiene que seleccionar el Checkbox **Integrar Extensiones** y una vez seleccionado se activan los botones **Adicionar** y **Eliminar**.

Cuando presiones el botón **Adicionar** aparecerá una ventana que le permitirá seleccionar nuevas extensiones para LibreOffice.

Dos extensiones no pueden tener el mismo nombre y tienen que ser obligatoriamente en formato .oxt porque las extensiones en LibreOffice son de este tipo de formato.

El botón **Eliminar** le permitirá al usuario eliminar la extensión seleccionada de la lista.

Prototipo de interfaz:



Historia de Usuario	
Número: HU_4	**Nombre Historia de Usuario:** Integrar, adicionar y eliminar plantillas al código fuente LibreOffice.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Alta	**Puntos Reales:** 0.1 semana
Descripción: Permitirá adicionar y eliminar al código fuente de LibreOffice nuevas plantillas.	

Observaciones:

Para integrar una nueva plantilla el usuario tiene que seleccionar el Checkbox **Integrar Plantillas** y una vez seleccionado se activan los botones **Adicionar** y **Eliminar**.

Cuando presiones el botón **Adicionar** aparecerá una ventana que le permitirá seleccionar plantillas para LibreOffice.

Los tipos de plantillas que el usuario podrá adicionar al código fuente de LibreOffice son:

- ✓ Plantillas para LibreOffice Impress (*.otp).
- ✓ Plantillas para LibreOffice Calc (*.ots).
- ✓ Plantillas para LibreOffice Writer (*.ott).

El botón **Eliminar** le permitirá al usuario eliminar la plantilla seleccionada de la lista.

Prototipo de interfaz:

Historia de Usuario	
Número: HU_5	**Nombre Historia de Usuario:** Integrar, adicionar y eliminar tipografía al código fuente LibreOffice.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Alta	**Puntos Reales:** 0.1 semana
Descripción: Permitirá adicionar y eliminar tipografías al código fuente de LibreOffice.	

Observaciones:

Para integrar una nueva tipografía el usuario tiene que seleccionar el Checkbox **Integrar Tipografía** y una vez seleccionado se activan los botones **Adicionar** y **Eliminar**.

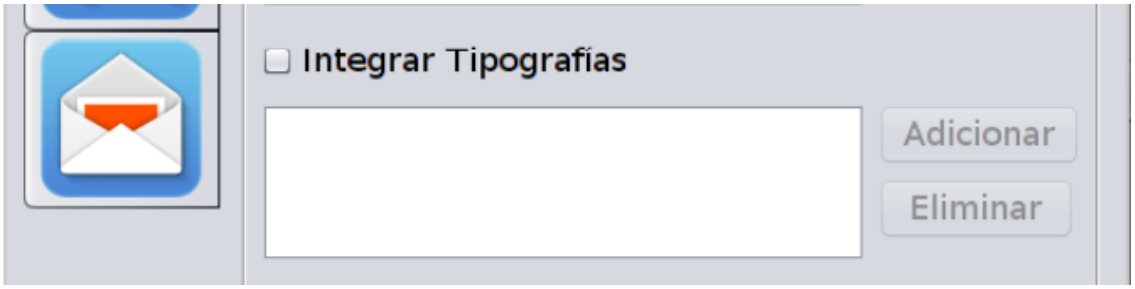
Cuando presiones el botón **Adicionar** aparecerá una ventana que le permitirá seleccionar plantillas para LibreOffice.

Los tipos de tipografías que el usuario podrá adicionar al código fuente de LibreOffice son:

- ✓ TrueType (*.ttf).
- ✓ OpenType (*.otf).
- ✓ PostScript tipo-1 (*.pfb).

El botón **Eliminar** le permitirá al usuario eliminar la tipografía seleccionada de la lista.

Prototipo de interfaz:



Historia de Usuario	
Número: HU_6	**Nombre Historia de Usuario:** Seleccionar código fuente de LibreOffice.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá al usuario seleccionar el código fuente de LibreOffice.	

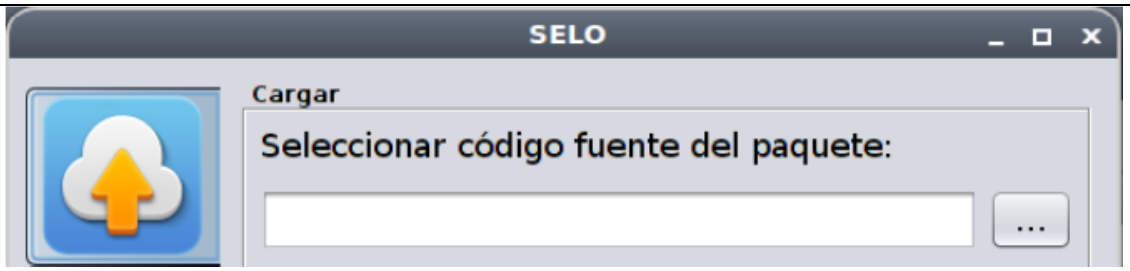
Observaciones:

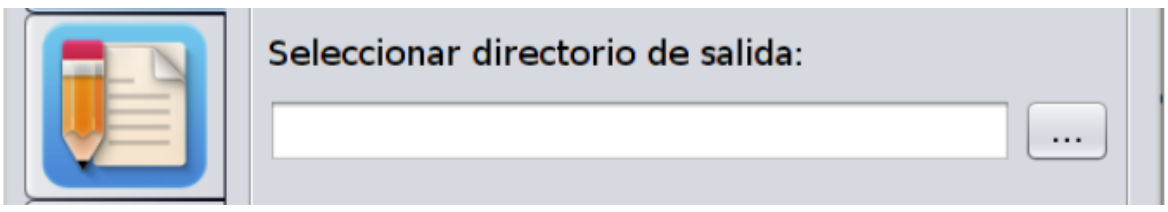
En la opción **Cargar** es obligatorio seleccionar el código fuente de LibreOffice, para poder realizar la compilación y empaquetar el mismo. El código fuente puede encontrarse en una carpeta en la computadora origen o en un servidor.

Solo se puede seleccionar una dirección válida o de lo contrario mostrará un error: "Parámetro incorrecto. El parámetro origen del código fuente es obligatorio y debe seleccionar un directorio".

Ejemplo: D:\Códigos\core

Prototipo de interfaz:



Historia de Usuario	
Número: HU_7	**Nombre Historia de Usuario:** Seleccionar el directorio de salida.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá seleccionar el directorio de salida que contendrá el nuevo código de LibreOffice compilado y empaquetado.	
Observaciones: Para seleccionar el directorio de salida el usuario tiene que seleccionar el botón **Cargar**. En él se encuentra la opción "**Seleccionar directorio de salida:**". Cuando se selecciona esta opción, aparece una ventana, que permitirá escoger la carpeta donde se copiará el código fuente compilado. Se recomienda realizar la copia dentro de una sub-carpeta para que una vez compilado el código fuente de LibreOffice, pase al proceso de empaquetamiento (generación de los .deb) los genere dentro de una carpeta aparte del código fuente.	
Prototipo de interfaz:  	

Historia de Usuario	
Número: HU_8	**Nombre Historia de Usuario:** Adicionar nombre, versión, distribución, arquitectura, autor, correo electrónico, descripción del paquete.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá insertar la información del paquete.	

Observaciones:

Es obligatorio adicionar nombre del paquete porque una vez compilado el código fuente con las nuevas transformaciones, se generarán los .deb con el nombre que el usuario adicionó en la caja de texto **Nombre del paquete**.

Si el usuario no llena este campo aparecerá un mensaje: "Parámetro incorrecto. El parámetro nombre del paquete es obligatorio."

Si el usuario llena el campo tiene que conocer que solo puede introducir nombres al paquete sin caracteres especiales. En caso contrario aparecerá un mensaje: "Parámetro incorrecto. El parámetro nombre del paquete solo permite letras sin caracteres especiales."

Ejemplo válido: libreoffice

Es obligatorio adicionar la versión del paquete porque una vez compilado el código fuente con las nuevas transformaciones, se generarán los .deb con el número de la versión que el usuario adicionó en la caja de texto **Versión**.

Si el usuario no llena este campo aparecerá un mensaje: "Parámetro incorrecto. El parámetro versión es obligatorio."

Si el usuario llena el campo tiene que conocer que solo puede introducir una versión válida. En caso contrario aparecerá un mensaje: "Parámetro incorrecto. El parámetro versión del paquete debe tener el siguiente formato: 4.1-1cesol0".

Es obligatorio adicionar la distribución de *software* porque una vez compilado el código fuente con las nuevas transformaciones, se generarán los .deb con el nombre de la nueva distribución que el usuario adicionó en la caja de texto **Distribución**.

Si el usuario no llena este campo aparecerá un mensaje: "Parámetro incorrecto. El parámetro distribución es obligatorio."

Si el usuario llena el campo tiene que conocer que solo puede introducir números sin caracteres especiales. Ejemplo válido: 2014

El usuario tendrá la posibilidad de seleccionar la arquitectura para el empaquetado que se va a realizar, las arquitecturas que podrá seleccionar son:

"Todas", "Arquitectura actual", "alpha", "amd64", "armel", "armhf", "hppa", "i386", "ia64", "mips", "mipsel", "powerpc", "powerpcspe", "ppc64", "s390", "s390x", "sparc", "kfreebsd-amd64", "kfreebsd-i386".

Donde la arquitectura “Todas” posibilita que el programa a empaquetar no dependa de la arquitectura del sistema. Se refiere a programas hechos en lenguajes interpretados o independientes de la plataforma, como Python o Java. El resultado será un paquete binario cuyo nombre acabará en *_all.deb*.

“Dependiente de la arquitectura” posibilitará el programa a empaquetar depende de la arquitectura del sistema y debe ser compilado en todas las arquitecturas compatibles. Habrá un archivo .deb para cada arquitectura (*_i386.deb* por ejemplo).

Si el usuario no selecciona una arquitectura específica, por defecto se seleccionará “Todas”.

Es obligatorio adicionar el autor porque una vez compilado el código fuente con las nuevas transformaciones, se generarán los .deb con el nombre del autor que el usuario adicionó en la caja de texto **Autor**.

Si el usuario no llena este campo aparecerá un mensaje: "Parámetro incorrecto. El parámetro autor es obligatorio."

Si el usuario llena el campo tiene que conocer que solo puede introducir letras y espacios. En caso contrario aparecerá un mensaje: "Parámetro incorrecto. El parámetro autor del paquete solo permite letras y espacios.". Ejemplo válido: Juan Pérez.

Es obligatorio adicionar el correo electrónico porque una vez compilado el código fuente con las nuevas transformaciones, se generarán los .deb con el correo electrónico del autor que el usuario adicionó en la caja de texto **Correo electrónico**.

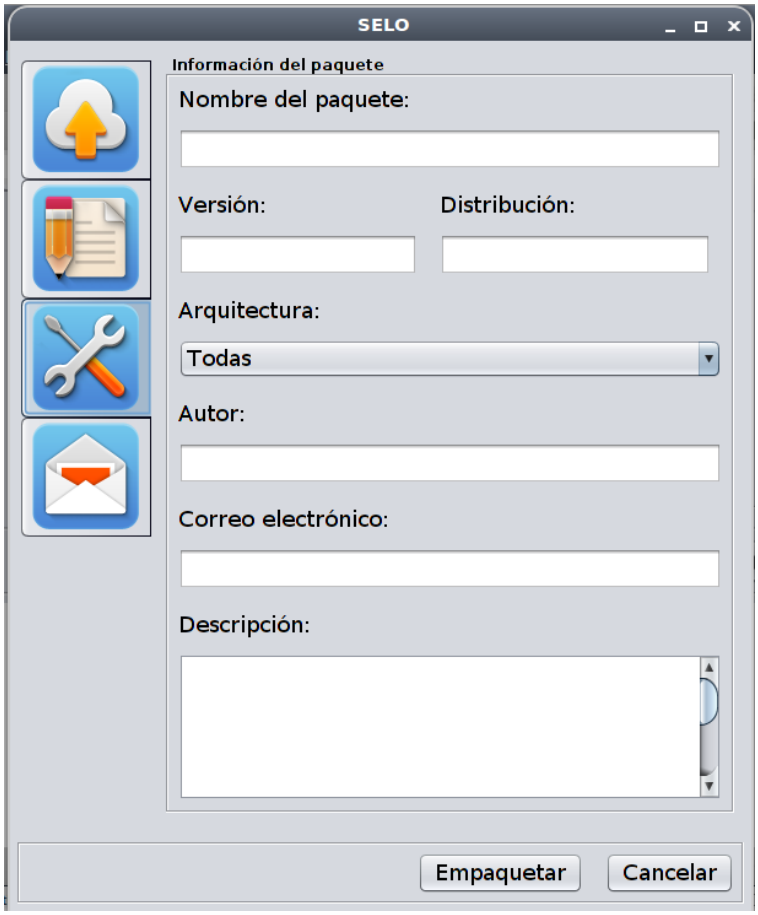
Si el usuario no llena este campo aparecerá un mensaje: "Parámetro incorrecto. El parámetro correo electrónico es obligatorio."

Si el usuario llena el campo tiene que conocer que solo puede introducir una cuenta de correo válida. En caso contrario aparecerá un mensaje: "El parámetro correo del autor no es válido.". Ejemplos válidos: juan@uci.cu o jua@estudiantes.uci.cu

Es obligatorio adicionar la descripción porque una vez compilado el código fuente con las nuevas transformaciones, se generarán los .deb con la breve descripción que el usuario adicionó en la caja de texto **Descripción**.

Si el usuario no llena este campo aparecerá un mensaje: "Parámetro incorrecto. El parámetro descripción es obligatorio."

Prototipo de interfaz:



Historia de Usuario	
Número: HU_9	**Nombre Historia de Usuario:** Mostrar y finalizar el proceso de empaquetamiento.
Modificación de Historia de Usuario Número: ninguna	

Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá mostrar el proceso actual por el cual está pasando el sistema.	
Observaciones: Para mostrar el proceso de empaquetamiento el usuario tiene que haber cargado, personalizado y llenado toda la información necesaria para una buena compilación y generación de los nuevo .deb. Una vez realizado lo anterior descrito solo tiene que presionar el botón **Empaquetar** y aparecerá una ventana indicando los pasos del proceso. Para finalizar el proceso de empaquetamiento, el usuario con solo presionar el botón **Finalizar,** automáticamente se detendrá el proceso de empaquetado.	
Prototipo de interfaz:	

Historia de Usuario	
Número: HU_10	**Nombre Historia de Usuario:** Mostrar información del sistema.
Modificación de Historia de Usuario Número: ninguna	
Usuario: todos	**Iteración Asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 0.1 semana
Riesgo en Desarrollo: Media	**Puntos Reales:** 0.1 semana
Descripción: Permitirá mostrar la información del sistema.	
Observaciones: Esta ventana mostrará si terminó correctamente el proceso de copiar el código fuente, integrarle extensión, plantillas o tipografías, y el tiempo de duración del sistema en empaquetar. Si todos los procesos fueron exitosos entonces estos paquetes están listos para utilizar en la migración.	
Prototipo de interfaz:	

Anexo 3: Tareas de Ingeniería.

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** HU-01
Nombre Tarea: Copiar el código fuente de LibreOffice.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 3/02/2014	**Fecha Fin:** 4/02/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Para copiar el código fuente de LibreOffice se implementa el método copiarDirectorio().	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** HU-01
Nombre Tarea: Compilar para generar los binarios de ejecución.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 4/02/2014	**Fecha Fin:** 16/02/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Compilar para generar los binarios de ejecución se utiliza la clase BufferedReader la cual es utilizada para la lectura de archivos.	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** HU-02
Nombre Tarea: Utilizar la clase File.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 16/02/2014	**Fecha Fin:** 17/02/2014

Programador Responsable: Juan Carlos Sacasas Cáceres
Descripción: La clase File es utilizada para el tratamiento de archivos.

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario:** HU-02
Nombre Tarea: Implementar el método Empaquetar para generar los paquetes .deb.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 17/02/2014	**Fecha Fin:** 25/02/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Para implementar el método Empaquetar para generar los paquetes .deb se utiliza la clase ProcessBuilder() para realizar las llamadas al sistema.	

Tarea de Ingeniería	
Número Tarea: 5	**Número Historia de Usuario:** HU-03
Nombre Tarea: Utilizar la clase JFileChooser.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 26/02/2014	**Fecha Fin:** 27/02/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: JFileChooser provee un mecanismo simple para que el usuario escoja un archivo.	

Tarea de Ingeniería	
Número Tarea: 6	**Número Historia de Usuario:** HU-04
Nombre Tarea: Utilizar la clase JOptionPane.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*

Fecha Inicio: 28/02/2014	**Fecha Fin:** 1/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: La clase JOptionPane facilita fortalecer una estándar de ventanas de diálogos que incita a los usuarios para un valor o les muestra información de algo.	

Tarea de Ingeniería	
Número Tarea: 7	**Número Historia de Usuario:** HU-05
Nombre Tarea: Diseñar la interfaz seleccionar código.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 1/03/2014	**Fecha Fin:** 3/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Las propiedades con los jButton, facilitará el tipo de acción, trabajar con la dimensiones, color de fondo, icono y texto.	

Tarea de Ingeniería	
Número Tarea: 8	**Número Historia de Usuario:** HU-06
Nombre Tarea: Diseñar la interfaz seleccionar directorio.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 3/03/2014	**Fecha Fin:** 4/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Las propiedades con los jButton, facilitará el tipo de acción, trabajar con la dimensiones, color de fondo, icono y texto.	

Tarea de Ingeniería

Número Tarea: 9	**Número Historia de Usuario:** HU-07
Nombre Tarea: Utilizar la clase Matcher.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 4/03/2014	**Fecha Fin:** 5/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Un motor que realiza las operaciones de los partidos en una secuencia de caracteres mediante la interpretación de un Patrón.	

Tarea de Ingeniería	
Número Tarea: 10	**Número Historia de Usuario:** HU-07
Nombre Tarea: Utilizar la clase Pattern.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 5/03/2014	**Fecha Fin:** 6/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Esta clase es utilizada para realizar una representación compilada de una expresión regular.	

Tarea de Ingeniería	
Número Tarea: 11	**Número Historia de Usuario:** HU-08
Nombre Tarea: Utilizar la clase ProcessBuilder.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 7/03/2014	**Fecha Fin:** 8/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	

Descripción: ProcessBuilder se usa para crear procesos del sistema operativo y llamadas al sistema.

Tarea de Ingeniería	
Número Tarea: 12	**Número Historia de Usuario:** HU-08
Nombre Tarea: Utilizar la clase Timer.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 8/03/2014	**Fecha Fin:** 9/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: Dispara una o más acciones de eventos a intervalos especificados.	

Tarea de Ingeniería	
Número Tarea: 13	**Número Historia de Usuario:** HU-08
Nombre Tarea: Utilizar la clase Calendar.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 10/03/2014	**Fecha Fin:** 11/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: La clase Calendar es una clase abstracta que proporciona métodos para convertir entre un instante específico en el tiempo y un conjunto de campos del calendario.	

Tarea de Ingeniería	
Número Tarea: 14	**Número Historia de Usuario:** HU-08
Nombre Tarea: Estudiar y probar la clase Date.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*

Fecha Inicio: 11/03/2014	**Fecha Fin:** 13/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: La clase Date representa un instante específico en el tiempo, con una precisión de milisegundos.	

Tarea de Ingeniería	
Número Tarea: 15	**Número Historia de Usuario:** HU-08
Nombre Tarea: Utilizar la clase GraphicsEnvironment.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 13/03/2014	**Fecha Fin:** 17/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: La clase GraphicsEnvironment describe la colección de objetos GraphicsDevice y objetos de fuente disponibles para Java (tm) la aplicación en una plataforma concreta.	

Tarea de Ingeniería	
Número Tarea: 16	**Número Historia de Usuario:** HU-08
Nombre Tarea: Utilizar la clase IOException.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 17/03/2014	**Fecha Fin:** 18/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: La clase IOException se usa para realizar señales que se ha producido en una excepción de E / S de una cierta clase. Esta clase es la clase general de excepciones producidas por las operaciones de E / S o sea fallidas o interrumpidas.	

Tarea de Ingeniería	
Número Tarea: 17	**Número Historia de Usuario:** HU-09
Nombre Tarea: Utilizar el método actionPerformed().	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 18/03/2014	**Fecha Fin:** 19/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: El método actionPerformed() permite mostrar el tiempo de duración.	

Tarea de Ingeniería	
Número Tarea: 18	**Número Historia de Usuario:** HU-10
Nombre Tarea: Implementar el método EjecutarPaso().	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** *1 semana*
Fecha Inicio: 19/03/2014	**Fecha Fin:** 20/03/2014
Programador Responsable: Juan Carlos Sacasas Cáceres	
Descripción: El método EjecutarPaso() permite ejecutar los pasos del proceso cuando se encuentra en ejecución. Para ello hace el llamado de los métodos que se encuentran en la clase controladora.	

Anexo 4: Estándar de codificación utilizado.

1. Deben evitarse los ficheros de gran tamaño que contengan más de 1000 líneas.
2. Los métodos deben agruparse por funcionalidad en lugar de agruparse por ámbito o accesibilidad, por ejemplo un método privado puede estar situado entre dos métodos públicos, el objetivo es desarrollar código fácil de leer y comprender.
3. Como norma general se establecen 4 caracteres como unidad de sangría. Los entornos de desarrollo integrado (IDE) más populares, tales como Eclipse o Netbeans que es el

que se utiliza para el desarrollo de la herramienta, incluyen facilidades para formatear código Java.

4. La longitud de línea no debe superar los 80 caracteres por motivos de visualización e impresión.
5. Cuando una expresión ocupe más de una línea esta se podrá romper o dividir en función de los siguientes criterios:

- Tras una coma.
- Antes de un operador.
- Alinear la nueva línea con el inicio de la expresión al mismo nivel que la línea anterior.

Ejemplos:

```
copiarArchivoBuscarRemplazar(String source_file, String destination_file,
                                        String toFind, String toReplace) {
if (source_code.getPath().endsWith(".otp")
        || source_code.getPath().endsWith(".ots")
        || source_code.getPath().endsWith(".ott")) {
```

6. Los comentarios deben contener el siguiente formato:

- Comentarios de bloque, permiten la descripción de ficheros, clases, bloques, estructuras de datos y algoritmos.

```
/*
 * Esto es un comentario
 * de bloque
 */
```

- Comentarios de línea, son comentarios cortos localizados en una sola línea. Si ocupa más de una línea se utilizará un comentario de bloque.

```
/* Esto es un comentario de línea */
```

```
// Esto es otro comentario de línea
```

- Comentario al final de la línea.
- Comentario situado al final de una sentencia de código y en la misma línea.

```
int contador = 0; // Inicialización del contador

contador++; /* Incrementamos el contador */
```

7. Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente.

Ejemplo:

```
int idUnidad = 1;

String[] funciones = { "Administración", "Intervención", "Gestión" };
```

8. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso. La única excepción a esta regla son los índices de los bucles "for", ya que en Java pueden incluirse dentro de la propia sentencia "for".

Ejemplo:

```
public void unMetodo() {

 int contador = 0; // inicio del método

 ...

}

for (int i=0; contador<10; i++) {

int contador1 = 0;

...

}
```

10. Cada línea debe contener como máximo una sentencia.

Ejemplo:

```
int contador++;

int variable--;
```

11. Todas las sentencias de un bloque deben encerrarse entre llaves "{ ... }", aunque el bloque conste de una única sentencia. Esta práctica permite añadir código sin cometer errores accidentalmente al olvidar añadir las llaves.

Ejemplo:

```
if (condicion) {
 variable++;
}
```

12. La sentencia "try/catch" siempre debe tener el formato siguiente:

```
try {
        sentencias;
} catch (ClaseException e) {
        sentencias;
}
```

13. Los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúsculas. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúscula.

Ejemplo:

```
class Paquete

class PaqueteDestino
```

14. Las variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas.

15. Las variables nunca podrán comenzar con el carácter "_" o "$". Los nombres de variables deben ser cortos y sus significados tienen que expresar con suficiente claridad la función que desempeñan en el código.

Ejemplo:

```
Paquete paquete;

String nombre;
```

16. Use minúsculas para constantes nativas de Java, siempre use cadenas en mayúsculas para la definición de sus propias constantes.

Ejemplo:

```
boolean variable=true;

int CONSTANTE;
```

Anexo 5: Casos de Prueba de Aceptación.

Caso de Prueba de Aceptación	
Código Caso de Prueba: 03	**Nombre Historia de Usuario:** Integrar, adicionar y eliminar extensiones al código fuente LibreOffice.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad integrar, adicionar y eliminar extensiones al código fuente LibreOffice.	
Condiciones de Ejecución: Acceder a la vista Personalización.	
Entrada / Pasos de ejecución: 1. Se presiona el botón Personalización. 2. Seleccionar Integrar Extensiones. 3. Se presiona el botón Adicionar. 4. Se selecciona la extensión para LibreOffice. 5. Se selecciona de la lista una extensión y se presiona el botón Eliminar.	
Resultado Esperado: Debe aparecer una ventana llamada personalización que permita integrar, adicionar y eliminar	

extensiones.
Evaluación de la Prueba: Satisfactoria.

Caso de Prueba de Aceptación	
Código Caso de Prueba: 04	**Nombre Historia de Usuario:** Integrar, adicionar y eliminar plantillas al código fuente LibreOffice.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad integrar, adicionar y eliminar plantillas al código fuente LibreOffice.	
Condiciones de Ejecución: Acceder a la vista Personalización.	
Entrada / Pasos de ejecución: 1. Se presiona el botón Personalización. 2. Seleccionar Integrar Plantillas. 3. Se presiona el botón Adicionar. 4. Se selecciona la plantilla para LibreOffice. 5. Se selecciona de la lista una plantilla y se presiona el botón Eliminar.	
Resultado Esperado: Debe aparecer una ventana llamada personalización que permita integrar, adicionar y eliminar plantillas.	
Evaluación de la Prueba: Satisfactoria.	

Caso de Prueba de Aceptación	
Código Caso de Prueba: 05	**Nombre Historia de Usuario:** Integrar, adicionar y eliminar tipografía al código fuente LibreOffice.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad integrar, adicionar y eliminar tipografía al código fuente LibreOffice.	
Condiciones de Ejecución: Acceder a la vista Personalización.	

Entrada / Pasos de ejecución: 1. Se presiona el botón Personalización. 2. Seleccionar Integrar Tipografía. 3. Se presiona el botón Adicionar. 4. Se selecciona la tipografía para LibreOffice. 5. Se selecciona de la lista una tipografía y se presiona el botón Eliminar.
Resultado Esperado: Debe aparecer una ventana llamada personalización que permita integrar, adicionar y eliminar tipografía.
Evaluación de la Prueba: Satisfactoria.

Caso de Prueba de Aceptación	
Código Caso de Prueba: 06	**Nombre Historia de Usuario:** Seleccionar código fuente de LibreOffice.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad cargar código.	
Condiciones de Ejecución: Acceder a la vista Cargar.	
Entrada / Pasos de ejecución: 1. Abrir la herramienta (SELO). 2. Seleccionar código fuente del paquete. 3. Si se selecciona un directorio de código fuente no valido debe aparecer una ventana con un mensaje: "Parámetro incorrecto. El parámetro origen del código fuente es obligatorio y debe seleccionar un directorio". 4. Si se entra un directorio valido se presiona el botón abrir.	
Resultado Esperado: Debe aparecer una ventana que permita cargar el código fuente de LibreOffice.	
Evaluación de la Prueba: Satisfactoria.	

Caso de Prueba de Aceptación	
Código Caso de Prueba: 07	**Nombre Historia de Usuario:** Seleccionar

	directorio de salida.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad seleccionar directorio de salida.	
Condiciones de Ejecución: Acceder a la vista Cargar.	
Entrada / Pasos de ejecución: 1. Abrir la herramienta (SELO). 2. Seleccionar directorio de salida. 3. Si se selecciona un directorio de salida no valido debe aparecer una ventana con un mensaje: "Parámetro incorrecto. El parámetro directorio de salida es obligatorio y debe seleccionar un directorio". 4. Si se entra directorio de salida valido, se presiona el botón seleccionar.	
Resultado Esperado: Debe aparecer una ventana que permita seleccionar el directorio de salida.	
Evaluación de la Prueba: Satisfactoria.	

Caso de Prueba de Aceptación	
Código Caso de Prueba: 08	**Nombre Historia de Usuario:** Adicionar nombre, versión, distribución, arquitectura, autor, correo electrónico, descripción del paquete.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad información del paquete.	
Condiciones de Ejecución: Acceder a la vista Información del paquete.	
Entrada / Pasos de ejecución: 1. Se presiona el botón Información del paquete. 2. Se adiciona el nombre del paquete. 3. Se adiciona la versión. 4. Se adiciona la distribución. 5. Se selecciona la arquitectura. 6. Se adiciona el autor. 7. Se adiciona el correo electrónico. 8. Se adiciona una breve descripción del paquete. 9. Se presiona el botón Empaquetar.	

Resultado Esperado: Debe aparecer una ventana que permita llenar toda la información y descripciones obligatorias del paquete.
Evaluación de la Prueba: Satisfactoria.

Caso de Prueba de Aceptación	
Código Caso de Prueba: 09	**Nombre Historia de Usuario:** Mostrar y finalizar el proceso de empaquetamiento.
Nombre de la persona que realiza la prueba: Grettel Barrio Marshall.	
Descripción de la Prueba: Prueba a la funcionalidad mostrar vista del proceso de empaquetamiento.	
Condiciones de Ejecución: Acceder a la herramienta.	
Entrada / Pasos de ejecución: 1. Abrir la herramienta (SELO). 2. Seleccionar código fuente del paquete. 3. Seleccionar directorio de salida. 4. Presionar el botón Información del paquete. 5. Se adiciona el nombre del paquete. 6. Se adiciona la versión. 7. Se adiciona la distribución. 8. Se selecciona la arquitectura. 9. Se adiciona el autor. 10. Se adiciona el correo electrónico. 11. Se adiciona una breve descripción del paquete. 12. Se presiona el botón Empaquetar.	
Resultado Esperado: Debe aparecer una ventana llamada Empaquetando LibreOffice, que mostrar todos los paso por el cual está pasando la herramienta y mostrar un mensaje final: “Paquetes debian generados”.	
Evaluación de la Prueba: Satisfactoria.	

Glosario de términos

Asistente: Aplicación informática que guía al usuario inexperto en el manejo de un programa.

Automatizar: Aplicar máquinas o procedimientos automáticos en la realización de un proceso o en una herramienta.

Calendario: Sistema de representación del paso de los días, agrupados en unidades superiores como semanas, meses y años.

Chroot: Es una operación que cambia el aparente directorio raíz para el proceso actualmente en ejecución y sus hijos.

Código fuente: Instrucciones y expresiones de un programa escritas por el programador en un lenguaje determinado, no es ejecutable directamente por la computadora puede ser escrito en un editor de texto y guardado en un archivo que luego hay que convertir a código máquina para que la computadora “lo entienda”. Cuando el código fuente de un programa es de libre acceso se dice que es código abierto.

Compilación: Proceso durante el cual se traduce un lenguaje de alto nivel a un programa de lenguaje máquina.

Debuild: Fichero de procesamiento por lotes especializado creado por el proyecto Gentoo Linux para usarlo con el sistema de mantenimiento de *software*: Portage, es una forma automática de compilar e instalar *software*.

GNU/Linux: Es uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux con el sistema GNU.

Hardware: Término general para los artefactos físicos de una tecnología, puede aplicarse a los componentes físicos de un sistema de cómputo.

Microsoft Office: *Suite* ofimática de la empresa Microsoft, unos de los programas que puede incluir (de acuerdo con la versión) son los siguientes: Word, Excel, PowerPoint, Visio, Access, FrontPage.

LibreOffice: Es una *Suite* Ofimática libre, la cual fue creada como una bifurcación de la también libre Openoffice.org tras la adquisición de *Sun Microsystems* por parte de la

multinacional *Oracle Corporation* y la posterior cancelación del proyecto *OpenSolaris* por parte de la misma corporación. Actualmente se encuentra en la versión 4.2.0.4 y es la aplicación ofimática por defecto en muchas distribuciones Linux como *OpenSUSE*, Ubuntu, Fedora, Nova. Esta *suite* incluye aplicaciones tales como: Writer, Calc, Impress, Base, Math y Draw las cuales son de uso habitual en una oficina.

Ofimática: Es un acrónimo compuesto por los términos oficina e informática. El concepto hace referencia a la automatización de las actividades que se realizan en una oficina.

PDF: (Portable Document Format) Formato de Documento Portátil es un formato de archivo sencillo, tiene como objetivo representar al documento de forma independiente al periférico de reproducción, sea este una pantalla de ordenador o una impresora. Los archivos PDF pueden ser creados a través de aplicaciones como el Acrobat de la empresa Adobe.

PGP: (*Pretty Good Privacy*): Programa informático que provee privacidad criptográfica y autenticación, a menudo se utiliza para firmar, encriptar y desencriptar correos electrónicos para aumentar la seguridad de las comunicaciones por esta vía, fue creado por Philip Zimmermann en 1991.

***Suite* Ofimática:** Es un conjunto de programas de ordenador diseñados para el trabajo de oficina, las mismas incluyen procesador de texto, hoja de cálculo, gestión de base de datos, cliente de correo electrónico, agenda y administrador de presentaciones o diapositivas.

***Software*:** Término general fundamentalmente utilizado para datos almacenados digitalmente, tales como programas de computadoras y otros tipos de información leída y escrita por computadoras. El término fue acuñado de manera que contrastara con el término hardware más antiguo. En contraste con el hardware el *software* es intangible, significando que “no puede ser tocado”. En ocasiones el término se utiliza más estrechamente para referirse únicamente a aplicaciones de *software*.

Tarball: Formato de archivos ampliamente utilizado en entornos UNIX.

Terminal: Es una forma de acceder al sistema sin utilizar la interfaz gráfica, es decir, realiza todo tipo de tareas en formato texto. La forma de utilizar el sistema de este modo es mediante órdenes.