Universidad de las Ciencias Informáticas

Facultad 4

Herramienta para el proceso de empaquetamiento de actualizaciones en el Sistema de Gestión para el Ingreso a la Educación Superior

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

AUTOR:

Dayron Sagarra Barreras

TUTORES:

Ing. Jorge Luis Piña González

Ing. Daniel Díaz León

Ing. Ulises Fernández-Nespral Vázquez

Ing. Rachelys Cruz Rodríguez

La Habana, junio de 2018

"Año 60 de la Revolución"

Declaración de Autoría

Declaro que soy el único autor del trabajo “Herramienta para el proceso de empaquetamiento de actualizaciones en el Sistema de Gestión para el Ingreso a la Educación Superior” y autorizo a la facultad 4 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Autor:

_____
Dayron Sagarra Barreras

Tutores:

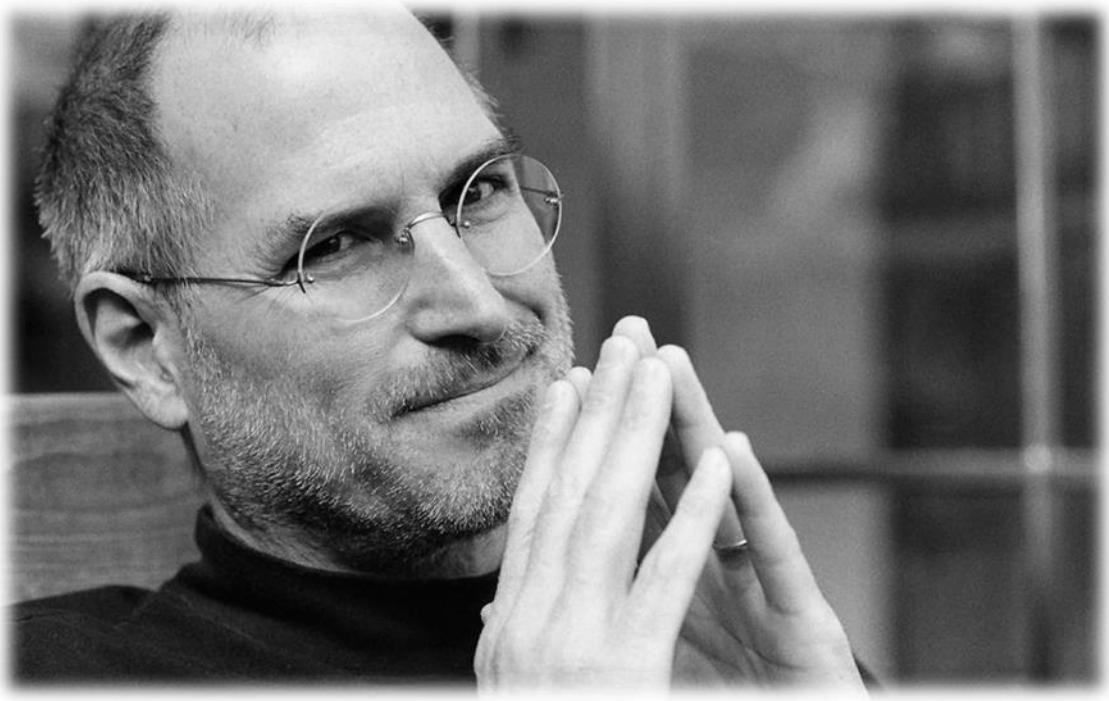
_____ _____
Ing. Jorge Luis Piña González Ing. Daniel Díaz León

_____ _____
Ing. Ulises Fernández-Nespral Vázquez Ing. Rachelys Cruz Rodríguez

Pensamiento

“A veces la vida te va a pegar en la cabeza con un ladrillo. Pero no pierdas la fe.”

Steve Jobs



Agradecimientos y Dedicatorias

En cuestión de agradecimientos, quiero primero que nada destacar la frase "Gracias Dios mío", la cual utilizamos todos en algún momento.

A mis padres, por el apoyo incondicional y emocional en toda la carrera.

A mi hermano Oriel Sagarra González.

Gracias a todos mis tutores por su dedicación y paciencia con este trabajo.

A todos los profesionales y profesores que radican en la facultad 4 y en toda la Universidad que día a día ponen de su empeño para que sus estudiantes lleguen a poner su nombre como Ingenieros. En especial a:

- *Reinier Fernández Coello*
- *Rafael Osvaldo Jiménez Rodríguez*
- *Karenia Donatien Goliath*
- *Marlon Rodríguez García*
- *Yunior Duque*

No podía faltar a todas estas amistades que he logrado con todo este tiempo, principalmente a todos los pertenecientes a mi grupo, que por nuestra unión todos hemos llegado a este lugar.

Un agradecimiento especial a Nadia Sánchez Lemus por su apoyo.

Agradecimientos y Dedicatorias

Este trabajo es dedicado a mi abuelo que ya no está entre nosotros, donde recuerdo que tenía razón en decir que todo, aunque parezca imposible solo basta con creer y actuar.

Dedicado también a mis padres, que se lo merecen por encima de cualquier cosa.

Resumen

El Ministerio de Educación Superior ha contratado a la Universidad de las Ciencias Informáticas para desarrollar un sistema de gestión para la informatización de los procesos a los que se enfrentan los estudiantes para integrar las filas universitarias; surge así, el Sistema de Gestión para el Ingreso a la Educación Superior. Este sistema necesita de un proceso de actualización informatizado con la finalidad de lograr un control de versiones para su mantenimiento. Realizar este proceso de forma manual ha traído como consecuencias algunas deficiencias que atrasan y dificultan el control efectivo de las actualizaciones. Por esta razón la presente investigación tiene como objetivo desarrollar una herramienta integradora que identifique los cambios de actualización del sistema y genere un paquete comprimido. Como guía se utilizó el Proceso Unificado Ágil en su variación UCI, generándose los artefactos necesarios. Se utilizó como lenguaje de programación Java, destacando el uso del Entorno de Desarrollo Integrado NetBeans, la herramienta para el modelado UML Visual Paradigm y el GitLab como controlador de versiones. Como resultado, la Herramienta para el Empaquetamiento de Actualizaciones de Sistemas, permite al cliente realizar comparaciones de forma efectiva y generar un paquete de actualización. Se realizaron pruebas a la solución propuesta para detectar las no conformidades y erradicarlas. Una vez concluidas se pudo apreciar que la herramienta presenta la calidad y funcionalidades requeridas por parte del cliente.

Palabras clave: actualización, empaquetamiento, versiones.

Índice

Índice

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC), son todos aquellos recursos, herramientas y programas que se utilizan para procesar, administrar y compartir la información mediante diversos soportes tecnológicos, tales como: computadoras, teléfonos móviles, televisores, así como reproductores portátiles de audio y video (Belloch Ortí, 2014).

Actualmente, el papel de las TIC en la sociedad es muy importante puesto que, ofrecen servicios como correo electrónico, búsqueda de información, banca en línea, comercio electrónico, descarga de música y video, entre otros. Con el desarrollo de la informatización en todas las esferas de la sociedad y el perfeccionamiento acelerado de las TIC, la mayoría de las empresas e instituciones necesitan informatizar los procesos de trabajo que desarrollan para ganar competitividad, eficiencia y tiempo. Muestra de esto, es el surgimiento de los Sistemas de Gestión de la Información (en lo adelante, SGI) con el objetivo de agilizar los procesos que se realizan en cada una de estas empresas e instituciones.

Según David Ingram, ***"****Los SGI son un conjunto de sistemas y procedimientos que recopilan información de una variedad de fuentes, la compilan y la presentan en un formato legible."* (Ingram, 2017).

Estos sistemas, son utilizados por la alta dirección de una empresa o institución para crear informes que les proporcionen una visión completa de toda la información que necesitan para tomar decisiones, que van desde pequeños detalles diarios hasta una estrategia de nivel superior. Los sistemas actuales de gestión de la información, se basan en gran medida en la tecnología para recopilar y presentar datos (Ingram, 2017)**.**

Actualmente, una de las esferas que se ha beneficiado con este desarrollo es la educación, aclarando que posee diferentes aplicaciones y sistemas con el objetivo de gestionar la información de las instituciones destinadas a la formación del estudiante, así como brindar funcionalidades para fortalecer métodos de aprendizajes y formas de estudios.

Cuba no está exenta a este proceso global, por lo que ha sido interés permanente del Estado el progreso de las TIC en la rama de educación, por tal motivo el Ministerio de Educación Superior (MES) lleva a cabo su proceso de informatización. Teniendo en

cuenta lo anterior la Universidad de las Ciencias Informáticas (UCI), específicamente el Centro de Tecnologías para la Formación (FORTES), se le ha dado la tarea de crear una plataforma que gestione el proceso de ingreso a la Educación Superior. Como resultado se obtuvo el producto Sistema de Gestión para el Ingreso a la Educación Superior (en lo adelante SIGIES), que tiene como objetivo principal facilitar el control en la información de los estudiantes próximos al proceso de ingreso a las universidades cubanas.

Al igual que algunas aplicaciones web, SIGIES necesita de la actualización de los archivos en desarrollo. Esta actualización se realiza con la finalidad de mantener su funcionamiento óptimo y así poder reparar posibles fallas, errores, vulnerabilidades y lo más importante, los posibles cambios constantes al que está sujeto dicho sistema debido a que cada año se tienen nuevos ingresos a las universidades cubanas.

Estos cambios pueden ser, en la colección de información organizada, o sea, en la base de datos y en versiones del propio sistema. Además, es preciso considerar que toda la información que se encuentra en él, define la trayectoria de los estudiantes que ingresan a la enseñanza superior, definiendo que dicha información es confidencial y delicada.

El modelo de despliegue propuesto para SIGIES, consiste en 3 servidores instalados en tres provincias: La Habana, Villa Clara y Santiago de Cuba; tiene como objetivo, minimizar los problemas de conectividad que puedan ocurrir en la red del MES, por lo que cada uno contiene una versión de SIGIES instalada, que podrá ser usada por los usuarios del MES que trabajan con el sistema.

El proceso de empaquetamiento de los archivos actualizados de SIGIES comienza cuando el cliente solicita que se realicen cambios en la colección de datos o cuando el sistema notifica que existe un error interno. Luego se analiza por parte del equipo de proyecto el documento de solicitud de cambio determinando las posibles soluciones a implementar.

Una vez desarrollada la solución, se debe realizar la actualización de los archivos. Esto consiste en que se tiene una copia de la versión del sistema instalada en los servidores y otra con la versión que integra los cambios ya resueltos. Luego se debe comparar ambas versiones, teniendo como posibles resultados las actualizaciones. En este proceso se utilizan los directorios y archivos del sistema, donde unas incorrectas manipulaciones de estos elementos pueden provocar deficiencias y dificultades, como son:

- Al utilizar los archivos y directorios del sistema es posible que exista el riesgo de que la actualización no sea aplicada a todos, pasando por alto alguno que sea considerado relevante.
- Resulta tedioso y difícil realizar el control de todos los cambios de manera visual, porque un sistema puede tener más de 60 000 archivos.
- Se pueden afectar los archivos debido a la compatibilidad y la seguridad, aclarando que una revisión no detallada puede provocar daños a cualquier archivo o directorio, incluso afectar las configuraciones internas del sistema; destacando la posible eliminación de cualquier código fuente considerado relevante.
- Los cambios identificados que se originan en el sistema no se pueden actualizar de forma directa en los servidores.
- El equipo de proyecto no tiene acceso a los servidores donde está instalado el sistema.

Por los motivos anteriormente se plantea el siguiente **problema científico**: ¿Cómo perfeccionar el proceso de empaquetamiento de las actualizaciones de SIGIES para disminuir las dificultades encontradas?

Lo cual determina como **objeto de estudio**: Proceso de empaquetamiento para la actualización del código fuente de sistemas web.

Para darle solución al problema científico se propone como **objetivo**: Desarrollar una herramienta integradora que identifique los cambios de actualización del sistema y genere un paquete comprimido.

Todo lo anterior precisa como **campo de acción**: Proceso de empaquetamiento del código fuente de SIGIES.

Para darle cumplimiento al objetivo general se plantean los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación a partir del estado del arte existente sobre el tema del empaquetado de directorios.
- Seleccionar las herramientas, tecnologías y metodología adecuadas para el desarrollo de una propuesta.
- Especificar los requisitos funcionales y no funcionales de la solución propuesta.

- Realizar el análisis y diseño de la solución en correspondencia con la metodología seleccionada.
- Implementar las funcionalidades.
- Validar la solución mediante pruebas de softwares.

Posibles resultados:

Una vez finalizado el presente trabajo, se tendrá como resultado una herramienta que permita identificar cambios en los archivos del sistema y genere un archivo comprimido para la actualización de SIGIES.

Para dar cumplimiento a los objetivos específicos planteados con anterioridad se proponen los siguientes **métodos científicos**:

Métodos teóricos:

- **Analítico- Sintético:** se utilizó para realizar el estudio bibliográfico acerca del objeto de estudio de la investigación, con el propósito de definir las características, herramientas y tecnologías de la propuesta de solución.
- **Histórico-lógico:** se empleó en el estudio de las diferentes formas de empaquetamiento de directorios con sus respectivos archivos.

Métodos empíricos:

- **Consulta de información en todo tipo de fuentes:** permitió la elaboración del marco teórico de la investigación.
- **Observación:** se utilizó para identificar algunas características de la propuesta de solución, como son herramientas, tecnologías y metodología a emplear.
- **Entrevista:** se utilizó para identificar las necesidades existentes en la línea de producción de SIGIES y de esta manera definir el objetivo al que estará dirigido la propuesta de solución.

El presente documento se estructura en 3 capítulos y los anexos, que incluyen los aspectos relacionados con el desarrollo de la aplicación que permitirá generar un paquete de actualizaciones para SIGIES.

Introducción

Capítulo 1: Fundamentación teórica: Se describen los aspectos y conceptos asociados al dominio del problema a resolver, siendo estos esenciales para entender el entorno del mismo. Se presenta el estado del arte de las técnicas implicadas en el objeto de estudio y el conjunto de tecnologías involucradas en el desarrollo de la propuesta y se realiza la justificación de las seleccionadas para la solución del problema.

Capítulo 2: Propuesta de solución: Se realiza una descripción de la propuesta de solución, así como toda la información relacionada con la metodología de desarrollo de software seleccionada en cuanto a las disciplinas de Requisitos, Análisis, Diseño e Implementación, a través de los objetivos específicos planteados para llevar a cabo el desarrollo de la solución que da lugar a la presente investigación.

Capítulo 3: Validación de la solución: En este capítulo, se llevan a la práctica las pruebas necesarias para la validación de la Herramienta para el proceso de empaquetamiento de actualizaciones del Sistema de Gestión para el Ingreso a la Educación Superior (SIGIES); sirviendo como guía lo planteado por la metodología AUP-UCI.

Capítulo 1: Fundamentación teórica

Capítulo 1: Fundamentación teórica

En este capítulo se aborda todo el marco teórico conceptual asociado al objeto de estudio. También se analizan las soluciones similares con la finalidad de establecer una correcta propuesta de solución. Por último, se describe la metodología de desarrollo de software, las herramientas y tecnologías que serán empleadas en la propuesta de solución.

1.1 Conceptos asociados al dominio del problema

Con el objetivo de mejorar la comprensión de la presente investigación, a continuación, se relacionan los principales términos relacionado con el objeto de estudio:

Proceso: Conjunto de las fases sucesivas de un fenómeno natural o de una operación artificial (Real Academia Española, 2017).

Actualizar: Poner al día datos, normas, reglas, etc. (Real Academia Española, 2017).

Proceso de actualización: El autor de la presente investigación, teniendo como referencia en que consiste proceso y actualización, obtiene que, un proceso de actualización es el conjunto de las fases sucesivas de una operación artificial que permite poner al día datos, normas, reglas etc. Más enfocado al tema de la presente investigación se puede decir, que es la sucesión de actos o acciones realizados con cierto orden, para sustituir un elemento antiguo por otro actual, logrando un cambio positivo, así sea para mejorar en recursos, contenido, funciones, código fuente o tiempo de ejecución de dicho proceso.

Paquete Debian: Archivo comprimido que cuenta con todos los elementos necesarios para instalar un programa en sistemas Linux de distribución Debian o derivados, con la extensión .deb. 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 que contienen uno información y el otro, los archivos que componen el programa empaquetado respectivamente, mientras que el último indica el número de versión del paquete DEB construido (Aoki, 2013).

Servidor: Un servidor o *server*, en el lenguaje informático, es un ordenador y sus programas, que están al servicio de otros ordenadores. El servidor atiende y responde

a las peticiones que le hacen los otros ordenadores, que son conocidos como los "clientes" del servidor (Significados.com, 2013).

Código fuente: En el contexto de la informática, el código fuente se define como el conjunto de líneas de textos, que son las directrices que debe seguir la computadora para realizar dicho programa**;** por lo que es en el código fuente, donde se encuentra escrito el funcionamiento para cualquier programa (Vendemia, 2014).

Empaquetar: Colocar convenientemente los paquetes dentro de bultos mayores (Real Academia Española, 2017).

Empaquetamiento: En el término de la informática es la posibilidad de tomar el código fuente de una aplicación y compilarlo, generando un “paquete” que no es más que un instalador para Debian en el formato .deb. es agrupar en un solo archivo varios archivos o directorios (Debian, 2016).

1.2 Estudio de soluciones similares

A continuación, se realiza un estudio de soluciones similares asociadas al problema planteado, para el cual se tuvo en cuenta aquellas herramientas con mayor demanda luego de consultar varias bibliografías en la web.

1.2.1 Herramientas existentes para la comparación de archivos

WinMerge

Es una herramienta de diferenciación y combinación de código abierto para Windows, puede comparar tanto carpetas como archivos, presentando las diferencias en un formato de texto visual, fácil de entender y controlar. Resulta muy útil para determinar lo que ha cambiado entre versiones de un proyecto y luego combinar los cambios entre dichas versiones. Además, se puede usar como una herramienta de diferenciación/combinación externa o una aplicación independiente (WinMerge, 2018).

Características

WinMerge tiene muchas características de asistencia que hacen que comparar, sincronizar y combinar sean lo más fáciles y útiles posibles:

General

- Microsoft Windows 2000/XP/2003/Vista/2008/7/8/2012.

- Manipula archivos de texto en formatos de Windows, Unix y Mac.
- Interfaz con pestañas.

Comparación de archivos

- Diferenciación y combinación visual de archivos de texto
- Editor flexible con resaltado de sintaxis, números de línea y ajuste de línea
- Resaltar diferencias dentro de las líneas
- El panel de diferencias muestra la diferencia actual en dos paneles verticales
- El panel de ubicaciones muestra un mapa de los archivos comparados
- Detección de líneas movidas. (Ver figura 1).

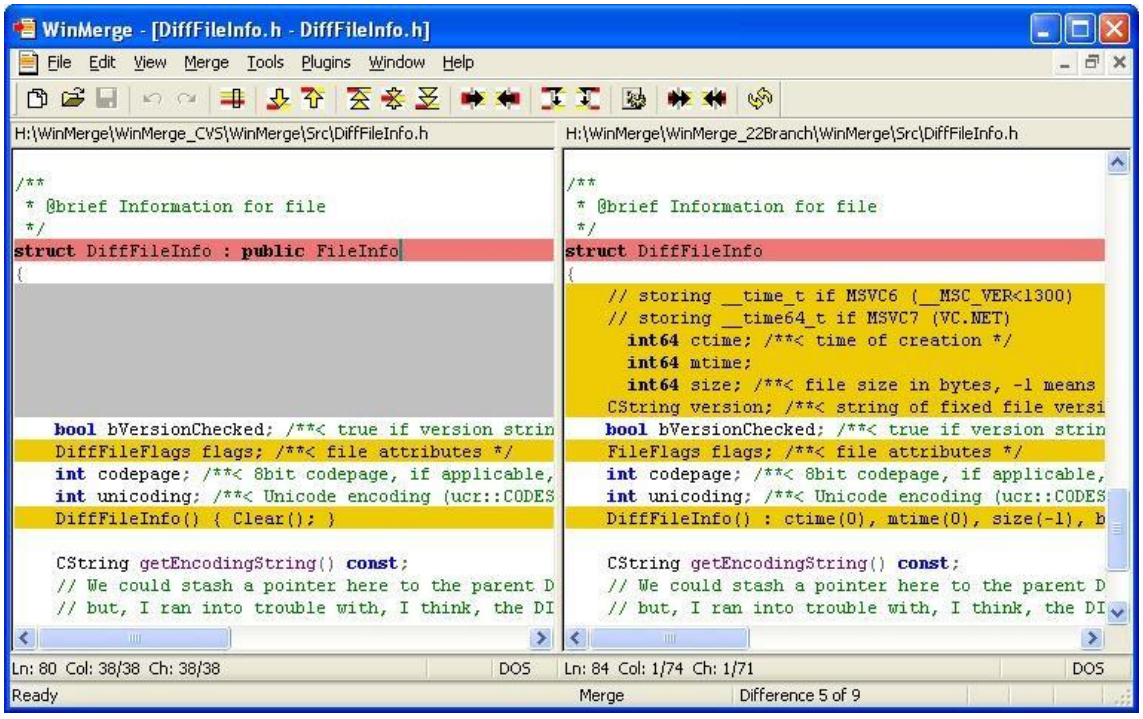


Figura 1: WinMerge

Meld

Hay varias herramientas para comparación de archivos o directorios en Linux, pero una de las más sencillas es Meld**.** Está en todos los repositorios de cualquier distribución y es una herramienta gráfica de código abierto (MeldMerge, 2016).

Capítulo 1: Fundamentación teórica

Características:

- Comparar la música en cualquier dispositivo electrónico con la carpeta de música en la computadora para mantenerlas iguales.
- Comparar las fotos de la cámara con las guardadas en la computadora.
- Comparar directorios en Linux.
- Se puede comparar no solo 2 archivos, sino que hasta 3.
- En el menú de “Cambios” se puede mezclar muy rápida y fácilmente los archivos o carpetas para que queden iguales. (Ver figura 2).

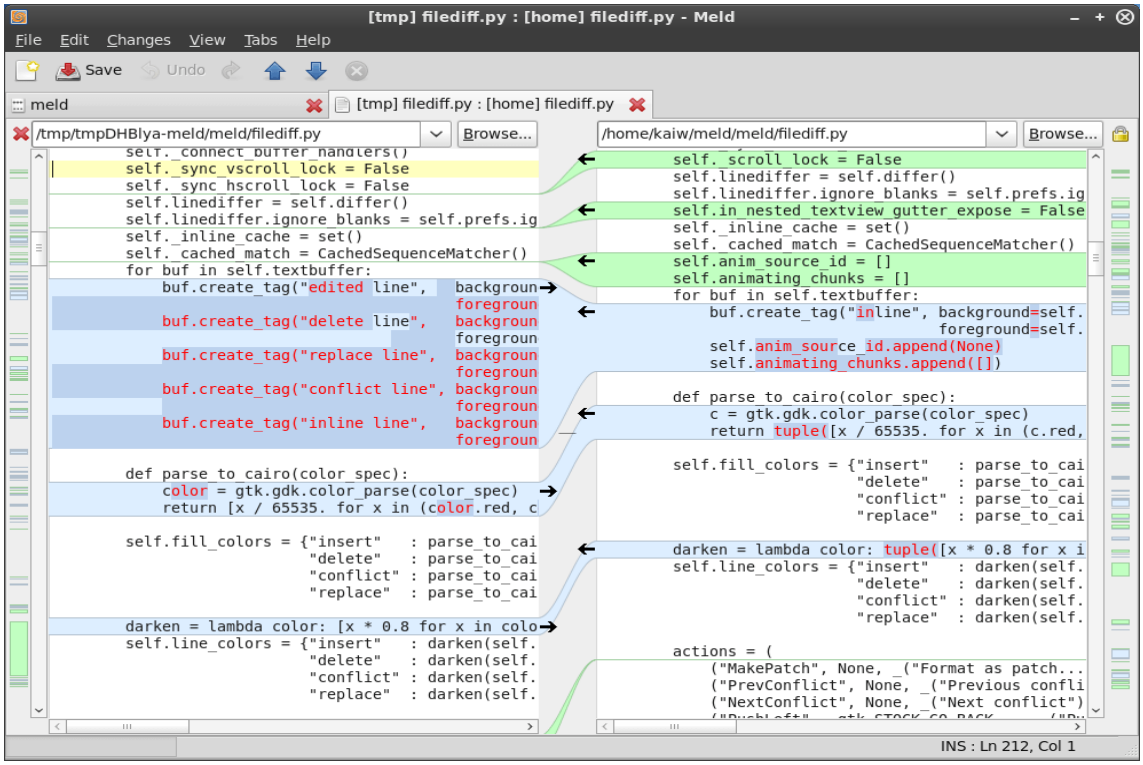


Figura 2: Meld

Luego de realizar el análisis de las herramientas anteriormente descritas se tiene que cumplen con la funcionalidad de comparación de archivos o directorios. Por otro lado, estas no satisfacen las necesidades para generar un paquete de actualizaciones, puesto que, no posibilitan realizar un ejecutable que tenga la capacidad de retener los resultados de la comparación antes mencionada.

1.2.2 Herramientas para la generación de paquetes Debian

dh_make

Es una herramienta para convertir un paquete de código fuente regular en uno formateado de acuerdo con los requisitos de la Política de Debian. dh_make se debe invocar dentro de un directorio que contenga el código fuente, que se debe llamar <packagename> - <version> (Debian, 2018). (Ver figura 3).

Figura 3: dh make

Debreate

Existen instrucciones sobre cómo realizar paquetes partiendo del código fuente, pero generalmente son bastante engorrosas y difíciles de comprender. Llevaría un tiempo largo entender el funcionamiento de dpkg[1] desde la consola.

Debreate es un programa que permite hacer esto de una forma muy sencilla. Realiza sus empaquetados para dicha generación del paquete Debian (AntumDeluge, 2018). (Ver figura 4).

[1] El comando **dpkg** es una herramienta para instalar, construir, eliminar los paquetes de Debian.

Capítulo 1: Fundamentación teórica

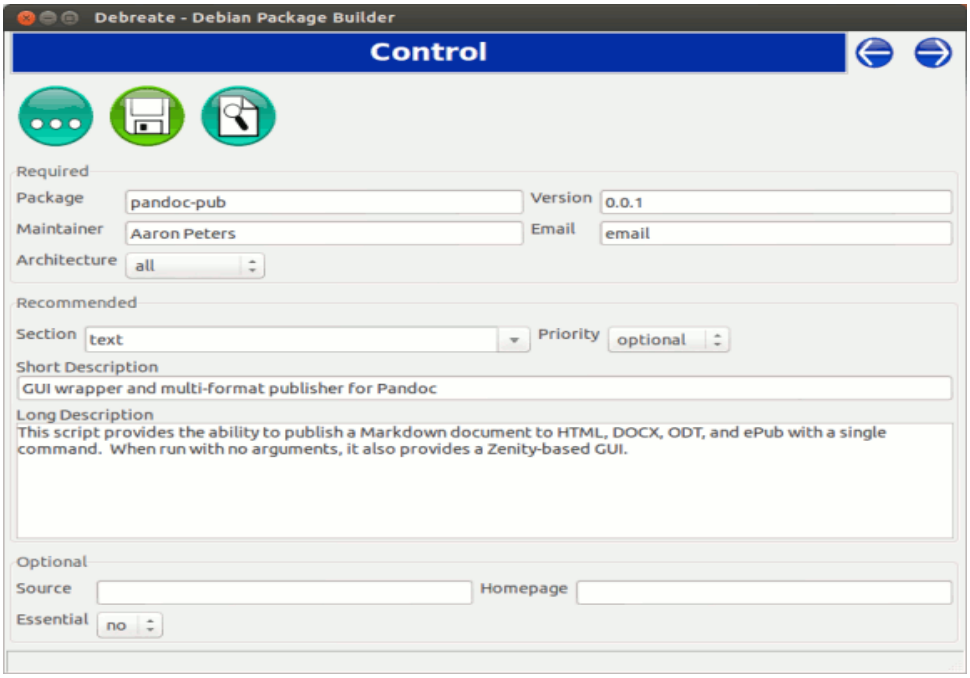


Figura 4: Debreate

Luego de realizar el estudio anterior sobre las herramientas para la generación de paquetes Debian se llegó a la conclusión de que estas no satisfacen las necesidades para la comparación de archivos o directorios.

1.3 Metodología de desarrollo de software

Una metodología de desarrollo de software consiste principalmente en hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo de software**.** Además, define quién debe hacer qué, cuándo y cómo debe hacerlo para obtener los distintos productos parciales y finales. Las metodologías se clasifican en dos grandes grupos: tradicionales y ágiles.

Al no existir una metodología de desarrollo software universal y como toda metodología debe ser adaptada a las características de cada proyecto, la UCI toma la iniciativa de hacer una variación de la metodología proceso unificado ágil (AUP por sus siglas en inglés de *Agile Unified Process*), de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la entidad (Rodríguez, 2015). Teniendo en cuenta que la presente investigación responde a uno de los proyectos de la red de centros de la UCI, se decide utilizar esta metodología para guiar el proceso de desarrollo de la propuesta de solución.

1.3.1 Variación de AUP para la UCI

La metodología de desarrollo de software Variación de AUP para la UCI tiene como propósito, que todos los proyectos de la UCI converjan hacia el uso de una sola metodología (Rodríguez, 2015). Esta variación de AUP para la UCI está formada por tres fases, (Inicio, Ejecución y Cierre) para el ciclo de vida de los proyectos de la universidad y propone las siguientes disciplinas, siendo estas: Modelado de Negocio, Requisitos, Análisis y diseño, Implementación, Pruebas Internas, Pruebas de Liberación y Pruebas de Aceptación. Las restantes disciplinas se cubren con las áreas de proceso que define el Modelo Integrado de Capacidad y Madurez para Desarrollo en su Versión 1.3, estas son Gestión de la configuración, Planeación de proyecto y Monitoreo y control de proyecto (Rodríguez, 2015).

Escenarios para la disciplina Requisitos

AUP en su variante UCI propone los siguientes escenarios:

- **Escenario No1:** proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS.

CUN + MC = CUS

- **Escenario No2:** proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS.

MC = CUS

- **Escenario No3:** proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP.

DPM + MC =DRP

- **Escenario No4:** proyectos que no modelen negocio solo pueden modelar el sistema con HU.

HU

Para el desarrollo de la solución propuesta, en la disciplina de Requisitos el autor de la presente investigación decide utilizar el escenario número 4. Esta selección se realiza, teniendo en cuenta, que el proyecto está bien definido, el cliente en todo momento estará junto al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Además, a pesar de la complejidad mostrada por el proyecto, este no es extenso, posibilitando el uso de las Historias de usuario.

1.4 Herramientas y tecnologías a utilizar

1.4.1 Herramientas de Ingeniería del Software Asistidas por Computadoras

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE, por sus siglas en inglés de *Computer Aided Software Engineering*) se pueden definir como un conjunto de programas y ayudas que dan asistencia a los analistas, desarrolladores e ingenieros de software, durante el ciclo de vida de un software, proporcionándole un aumento en su productividad y logrando un mayor ahorro de tiempo (Ambler, 2017). Para el modelado de los artefactos generados durante el desarrollo de la presente investigación se decide utilizar la herramienta Visual Paradigm.

Visual Paradigm: es una herramienta que soporta el ciclo de vida completo de desarrollo de un software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Permite representar gráficamente varios diagramas y facilita la generación de código. Es una herramienta multiplataforma, fácil de instalar y utilizar (Visual Paradigm, 2017).

Características:

- Soporte de UML[2].
- Diagramas de Procesos de Negocio.
- Ingeniería inversa.
- Modelo a código, diagrama a código.

[2] Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language)

- Diagramas de flujo de datos.
- Generación de bases de datos.
- Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Distribución automática de diagramas.

A partir de los elementos antes expuestos el autor de la presente investigación decide utilizar Visual Paradigm en su versión 8.0, teniendo en cuenta que esta herramienta propicia un conjunto de funcionalidades para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Además, se tuvo en cuenta que la UCI posee una licencia para su uso. Esta herramienta se utilizará para generar los artefactos que señala el escenario escogido.

1.4.2 Lenguajes de programación

Los lenguajes de programación son lenguajes artificiales que pueden utilizarse para definir una secuencia de instrucciones para su procesamiento por una computadora. Son herramientas que permiten crear programas y software. Los lenguajes de programación están formados por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura, además el significado de sus elementos y expresiones. Son utilizados para controlar el comportamiento lógico de una máquina y para expresar algoritmos con precisión. Estos lenguajes permiten ser leídos y escritos por personas y a su vez resultan independientes del modelo de computadora a utilizar. Existen varios lenguajes de programación tales como: Python, Java y C++ (Joyanes, 2013). En la presente investigación se decide utilizar el lenguaje Java.

Java: es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "*write once*, *run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra (Joyanes, 2013).

Java es la base para prácticamente todos los tipos de aplicaciones de red, además del estándar global para desarrollar y distribuir aplicaciones móviles, juegos, contenido basado en web y software de empresa. Con más de 9 millones de desarrolladores en

todo el mundo, Java le permite desarrollar, implementar y utilizar de forma eficaz interesantes aplicaciones y servicios (Expósito, 2006).

1.4.3 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés de *Integrated Development Environment*), es un medio de desarrollo que ha sido empaquetado como un programa de aplicación, generalmente está compuesto por un conjunto de herramientas de programación en las que se encuentra el compilador, editor de código, depurador y constructor de interfaces gráficas. Este proporciona un marco de trabajo amigable para diversos lenguajes de programación (Martínez, 2008).

NetBeans: es un entorno de desarrollo gratuito y de código abierto. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones web, o para dispositivos móviles. Da soporte a las siguientes tecnologías: Java, PHP, Groovy, C/C++ y HTML5. Además, puede instalarse en varios sistemas operativos: Windows, Linux o Mac OS (Netbeans.org, 2016).

Se selecciona para el desarrollo de la solución informática propuesta, la herramienta NetBeans en su versión 8.2, teniendo en cuenta que es libre, se integra perfectamente con el lenguaje de programación Java que es el definido para el desarrollo de la solución propuesta.

Componente Jmeld: es un componente integrador de código abierto que comprende la funcionalidad de efectuar las comparaciones de archivos y directorios, mostrando sus diferencias. Contiene el visual, que posibilita al usuario gestionar las diferencias encontradas (Meld, 2018).

1.4.4 Sistema de control de versiones

Un Sistema de Control de Versiones (VCS por sus siglas en inglés de *Version Control Systems*) que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo durante el desarrollo de un producto, de modo que se pueda recuperar versiones específicas más adelante. Permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo (Git, 2017). Siguiendo el paradigma de independencia tecnológica que se lleva a cabo en la UCI, se decide seleccionar como Sistema de Control de Versiones el GitLab.

Git: es un sistema de control de versiones distribuidas de código abierto y gratuito diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia. Es fácil de aprender y tiene una huella pequeña con un rendimiento increíblemente rápido. Además, es una herramienta gratuita para cualquier equipo de desarrollo, independientemente de su tamaño o del tipo de software que desarrolle. Se decide utilizar para lograr establecer el control de los avances durante el desarrollo de la propuesta de solución.

Conclusiones parciales

El análisis del marco teórico referencial asociado al proceso de actualización, permitió lograr una mejor comprensión del objeto de estudio de la presente investigación.

El estudio de las herramientas similares, posibilitó la obtención del resultado de que estas no satisfacen las necesidades para el proceso de empaquetamiento de actualizaciones en el SIGIES y es necesario desarrollar una nueva herramienta integradora que resuelva los problemas descritos anteriormente.

Además, el estudio de las herramientas y tecnologías asociadas al objeto de estudio de la presente investigación, permitió definir que, para el desarrollo de la propuesta de solución, es necesario utilizar:

- **Visual Paradigm 8.0:** Como herramienta para el modelado.
- **Java 8.0:** como lenguaje de programación.
- **NetBeans 8.2:** como Entorno de Desarrollo Integrado (IDE de desarrollo).
- **Scripts:** para disponer la ejecución, desde una interfaz en Java, de los comandos en consola.
- **Git 2.7.4:** como sistema de control de versiones.

Capítulo 2: Propuesta de solución

Capítulo 2 Propuesta de Solución

En el presente capítulo se realiza una descripción de la propuesta de solución, así como la información relacionada con la metodología de desarrollo de software seleccionada, en cuanto a las disciplinas de Requisitos, Análisis y Diseño e Implementación, a través de los objetivos específicos planteados para llevar a cabo el desarrollo de la solución. Seguidamente, se reflejan las necesidades del cliente. Se describen los patrones de diseños utilizados en el desarrollo incluyendo el estándar de codificación a seguir por el autor de la presente investigación.

2.1 Propuesta de solución

La solución que se propone tiene como objetivo permitir a los desarrolladores encargados de confeccionar el paquete de actualización de SIGIES, alcanzar mayor rapidez en el control y actualización del sistema, facilitando la confección de dicho paquete que contendrá todos los cambios detectados en las comparaciones. La herramienta debe ser capaz de realizar comparaciones tanto de directorios como de archivos y de esta forma obtener resultados que se evidenciarían en los cambios detectados por la herramienta. La solución debe brindar opciones para gestionar los resultados obtenidos, por ejemplo, copiar contenido detectado en un archivo o directorio hacia otro y viceversa. Una vez culminado este proceso la herramienta debe brindar la posibilidad de generar un paquete Debian, con los resultados finales, además de permitir incluso que se lleven a cabo las configuraciones pre y pos instalación del paquete generado.

2.2 Modelo de dominio

Con el objetivo de crear la base para analizar los requisitos del cliente durante el proceso de desarrollo de software, se decide realizar el modelo de dominio, en el cuál se relacionarán los conceptos que definen el proceso de empaquetamiento de actualizaciones en el Sistema de Gestión para el Ingreso a la Educación Superior.

2.2.1 Diagrama del modelo de dominio

A continuación, se refleja mediante el modelo de domino la relación entre los principales conceptos que engloban el dominio del problema de la presente investigación y sus relaciones. (Ver figura 5).

Capítulo 2: Propuesta de solución

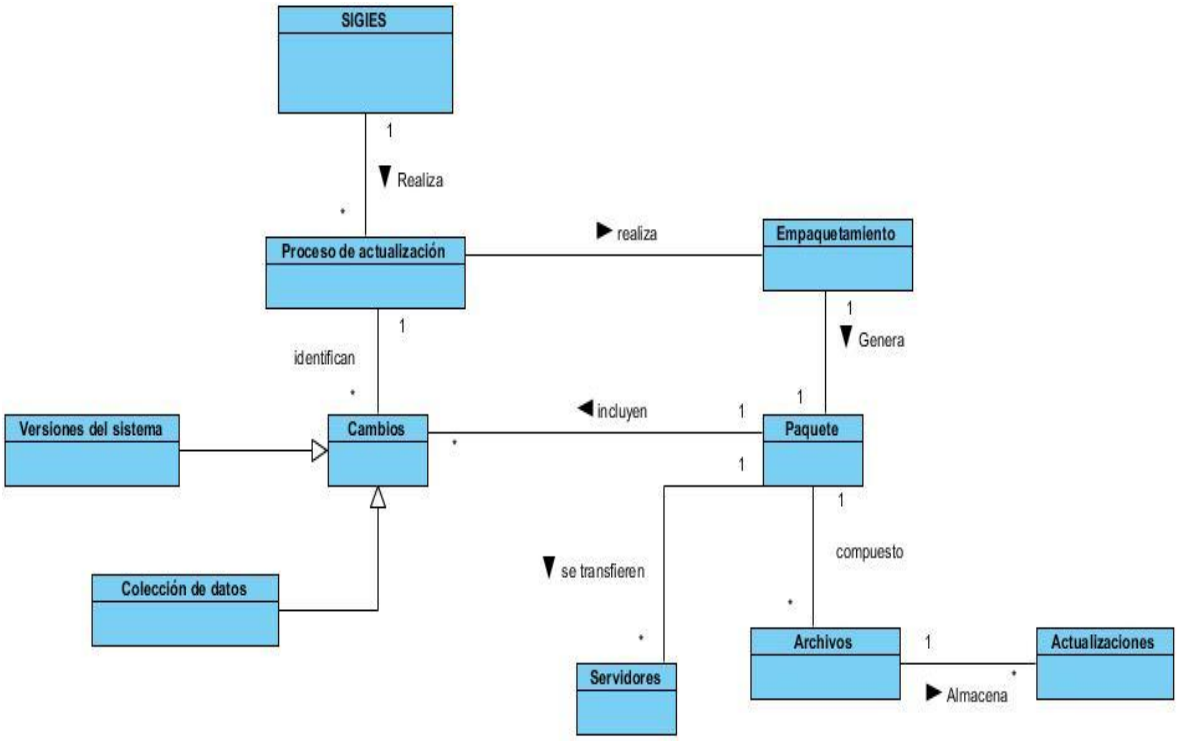


Figura 5: Modelo de Dominio (Elaboración propia)

- **SIGIES:** Sistema informático por el cual se maneja la información durante el proceso de ingreso a las universidades cubanas.
- **Proceso de actualización:** Sucesión de actos o acciones realizados con cierto orden, que se dirigen a remplazar un elemento antiguo por otro actual, para lograr un cambio positivo.
- **Cambios:** Transición que ocurre de un estado a otro.
- **Versiones del sistema:** Nombre, número o código dado a un sistema luego de hacerle cambios para mejorar su funcionamiento.
- **Colección de datos:** Conjunto de datos tabulados que responden a una única estructura en su almacenamiento.
- **Empaquetamiento:** Compilación del código fuente de una aplicación en un solo archivo.
- **Paquete:** Archivo comprimido que cuenta con todos los elementos necesarios para instalar un programa en sistemas Linux de distribución Debian o derivados.
- **Archivo:** Conjunto de bits que son almacenados en un dispositivo informático.
- **Servidores:** Son los ordenadores y programas que están al servicio de otros ordenadores
- **Actualizaciones:** Cambio positivo, así sea de contenido, funciones, código fuente o tiempo de ejecución de un recurso determinado.

Capítulo 2: Propuesta de solución

Una vez sentadas las bases del proceso de actualización en SIGIES y conociendo como se interrelaciona cada uno de los conceptos asociados al problema, se da paso a la disciplina de Requisitos, con el objetivo de facilitar un mecanismo apropiado para comprender lo que quiere el cliente, analizando sus necesidades y negociar una solución que transforme esos requisitos en un sistema operacional.

2.3 Requisitos

"Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema." (Pressman, 2010).

Los requisitos de software juegan un papel fundamental dentro de la etapa de desarrollo de un software. Tal es así que, una mala definición, especificación o administración de requisitos puede fácilmente llevar al fracaso total de lo que podría ser una buena solución informática. Por esta razón, en el presente epígrafe se especificará cuáles fueron las técnicas que se utilizaron para la definición de los requisitos, así como su especificación, a través de las técnicas de validación de los mismos, obteniéndose las bases para la fase de Análisis y Diseño.

2.3.1 Técnicas para el levantamiento de requisitos

Para identificar las necesidades del cliente se utilizan técnicas que permiten determinar de manera explícita y documentada los requisitos que tiene el cliente. Esta actividad es continua durante el ciclo de desarrollo y combina, en diferentes puntos, diversas técnicas de identificación para obtener la visión más completa de las necesidades del usuario final. Para la captura de los requisitos se utilizaron las siguientes técnicas:

Entrevista: Esta técnica es de gran utilidad para obtener información cualitativa, requiere seleccionar bien a los entrevistados para obtener la mayor cantidad de información en el menor tiempo posible. Es muy aceptada y permite acercarse al problema de una manera natural (Aristizábal Mejía, y otros, 2017)

Esta técnica fue aplicada a los desarrolladores de SIGIES a cargo de realizar la actualización de dicha aplicación. En el **Anexo 1**, se encuentra la guía de preguntas utilizada en el desarrollo de la entrevista.

Tormentas de ideas: Esta técnica se puede utilizar para identificar un primer conjunto de requisitos en aquellos casos donde no están muy claras las necesidades que hay que cubrir (Soulary, y otros, 2010).

Para la aplicación de esta técnica se realizó un encuentro entre los miembros del equipo de desarrollo y el cliente, donde ambas partes brindaron sus ideas en cuanto a la propuesta de solución. Los resultados de esta técnica están evidenciados en los acuerdos tomados en la minuta de reunión del encuentro. (Ver **Anexo 2**).

2.3.2 Requisitos funcionales

Los requisitos funcionales definen el funcionamiento del sistema. Estos varían según el tipo de sistema a implementar y de los posibles usuarios. Facilitan un entendimiento de los procesos a desarrollar, que se comprenda con profundidad el problema en cuestión y una mejor identificación de las funcionalidades que serán implementadas (Pressman, 2010).

La siguiente tabla muestra el listado de los requisitos funcionales identificados:

Tabla 1: Requisitos funcionales (Elaboración propia)

No.	Nombre	Descripción
RF1	Incluir ficheros.	La aplicación debe permitirle al usuario incluir los ficheros para poder hacer la comparación.
RF2	Comparar ficheros.	La aplicación debe garantizar la comparación de los dos ficheros.
RF3	Modificar datos de los ficheros.	La aplicación debe ser capaz de modificar los ficheros e insertar los cambios desde un fichero hacia el otro.
RF4	Configuraciones del paquete Debian.	La aplicación permite introducir las informaciones de las configuraciones internas del paquete debian, para crear un correcto empaquetado.
RF5	Insertar paquetes de dependencias.	La aplicación debe insertar los paquetes de dependencias, los cuales son considerados

		informaciones extras de otros paquetes necesarios para la instalación.
RF6	Ejecutar comandos de Configuración.	La aplicación debe ser capaz de crear archivos ejecutables, que almacena operaciones funcionales mediante comandos insertados por el usuario.
RF7	Ejecutar empaquetamiento.	La aplicación brindar la posibilidad de realizar el empaquetamiento de un directorio para crear el ejecutable con el formato .deb.
RF8	Incluir directorios.	La aplicación debe brindar la posibilidad de incluir directorios para realizar una comparación y un correcto empaquetamiento en uno de ellos.
RF9	Comparar directorios.	La aplicación debe garantizar la comparación de los dos directorios antes incluidos.
RF10	Modificar directorios.	La aplicación debe ser capaz de modificar el contenido de los directorios e insertar los cambios desde uno hacia el otro.

2.3.3 Requisitos no funcionales

Tabla 2: Requisitos no funcionales (Elaboración propia)

Clasificación	Descripción
Requisitos de Usabilidad	**RNF 1**- Las ventanas que proporcione la aplicación deben ser de fácil acceso y entendimiento para el usuario que la utilice.
Requisitos de interfaz y apariencia	**RNF 2**- Los menús presentados por la aplicación deben tener una buena interfaz de manera visual.

	RNF 3- Las interfaces que muestre la aplicación deben ajustarse al estilo de la aplicación.
Requisitos de software	**RNF 4**- Se debe tener instalada en la computadora la Máquina Virtual de Java en su versión 8 o superior. **RNF 5**- La aplicación debe ser compatible con el sistema operativo GNU/Linux y sus derivados.
Requisitos de hardware	**RNF 7**- Para ejecutar la aplicación, la computadora debe contar con las siguientes características: 2 GB de memoria RAM, equivalente o superior. Mínimo Dual Core a 2.6Ghz, recomendado Core i3 a 1.6 GHz.

2.3.4 Validación de requisitos

"La validación de requisitos demuestra que la definición de los mismos responde correctamente a las necesidades del cliente. Esta es una actividad fundamental, ya que un levantamiento de requisitos con errores, que no se detecten a tiempo conduce a resultados inesperados, provocan costos excesivos y gran pérdida de tiempo" (Pressman 2010).

Con el objetivo de asegurar que el software está de acuerdo con su especificación, se comprueba que el sistema cumple con los requisitos funcionales y no funcionales que se han especificado, donde se utiliza la siguiente técnica de validación de requisitos:

Revisiones técnicas formales de los requisitos: se realizaron revisiones formales de cada requisito, por parte del cliente y el autor de la presente investigación, validando que la interpretación de cada una de las descripciones no sea ambigua, ni presenten omisiones o errores. En el **Anexo 3**, se evidencian los acuerdos tomados en esta reunión, los cuales están recogidos en la minuta generada en este encuentro.

2.4 Historias de usuarios

Las Historias de usuario (HU) son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir

de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes, por lo que las historias de usuario pueden tener varios cambios a lo largo de un desarrollo sin afectarse el tiempo (Beck, 2002).

A continuación, se describen dos HU de prioridad alta en el desarrollo de la propuesta de solución, el resto se pueden consultar en los artefactos entregables generados según la metodología de desarrollo de software seleccionada. En el **Anexo 4** se definen las restantes historias de usuario.

Tabla 3: Historias de usuario #1 (Elaboración propia)

Número: 1	**Nombre del requisito:** Incluir ficheros
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 2 días
Riesgo en Desarrollo: Medio	**Tiempo Real:** 2 días
Descripción: La aplicación debe permitirle al usuario incluir los ficheros para poder hacer la comparación. 1. El usuario debe seleccionar la opción New que se encuentra en el menú Package. 2. El usuario selecciona la opción New que implicaría buscar nuevos ficheros, para realizar la comparación entre ellos.	
Observaciones: Debe permitir levantar el explorador de archivos para la búsqueda del fichero a incluir.	
Prototipo de interfaz:	

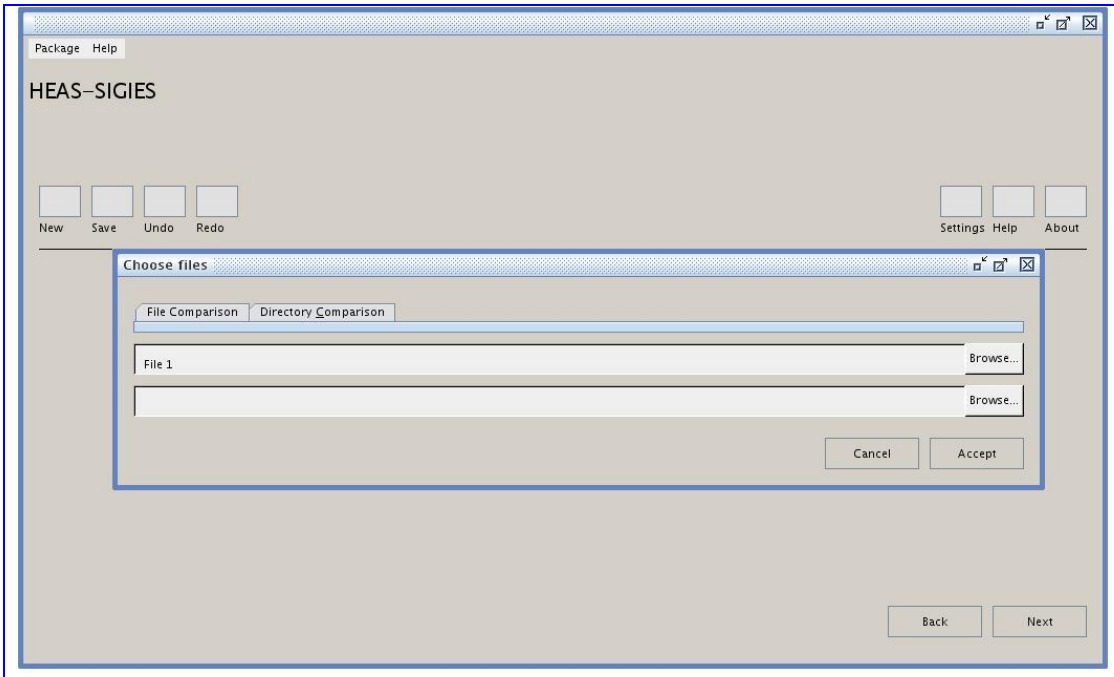


Tabla 4: Historias de usuario #2 (Elaboración propia)

Número: 9	**Nombre del requisito:** Comparar directorios.
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 4
Prioridad: Alta	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 2 días

Descripción: La aplicación debe ser capaz de comparar los Directorios seleccionados

- El usuario selecciona los directorios y seguidamente la opción Accept.
- El sistema muestra los resultados de comparar dos directorios seleccionados. Se visualizan los datos:

  -Nombre del directorio

  -Dirección de directorio

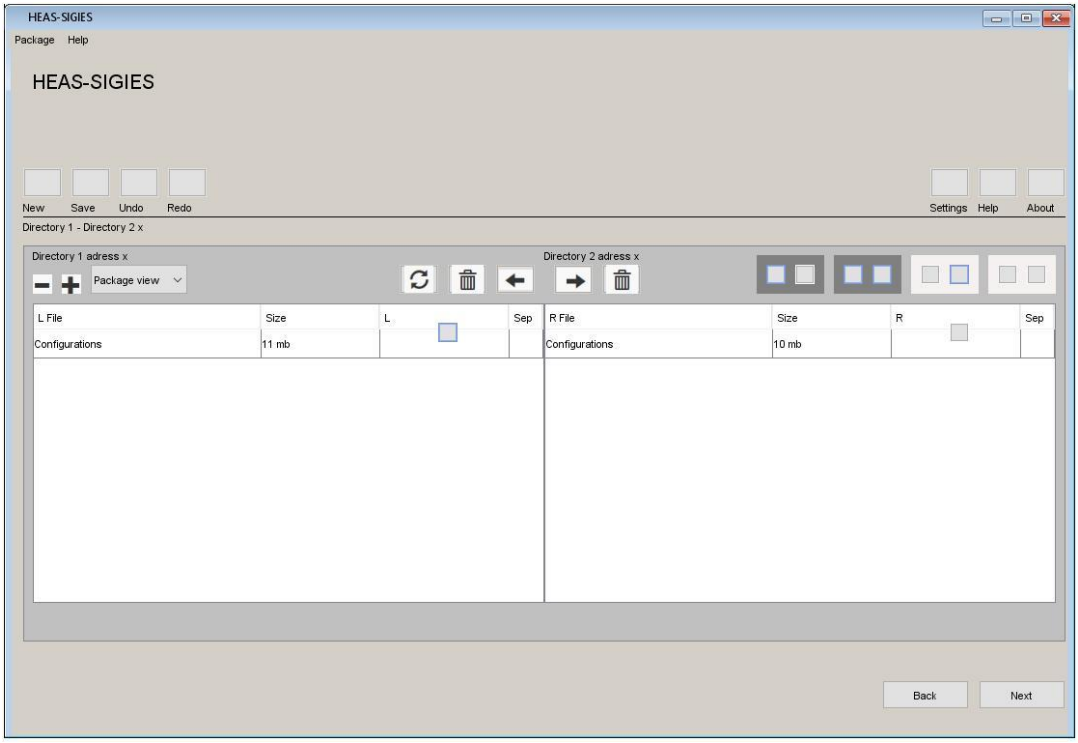
  -Contenido del directorio

  -L File (representa los ficheros del directorio Left)

  -Size (tamaño de los ficheros de los directorios)

  -R File (representa los ficheros del directorio Right)

  -L y R (Cambios en los ficheros Left y Right respectivamente)

-Sep (columna que permite separar un directorio de otro)
Observaciones: Deben existir dos directorios a comparar.
Prototipo de interfaz: 

Luego de realizar la representación de 2 de los requisitos identificados mediante las Historias de usuarios y con el objetivo de formalizar un lenguaje común el equipo de desarrollo, se da paso al diseño de la propuesta de solución, para lo cual se tuvo en cuenta los patrones de diseños que se muestran a continuación.

2.5 Patrones de diseño

Los desarrolladores con experiencia acumulan un conjunto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, son definidos como “Patrones” (Rojas, 2014).

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones

anteriores, otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias (Rojas, 2014).

2.5.1 Objetivos de los patrones

Los patrones de diseño pretenden (Visconti, y otros, 2013):

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.
- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones siempre, sólo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error.

Se pueden agrupar en dos grandes grupos; los Patrones Generales de Software para Asignación de Responsabilidades (GRASP por sus siglas en ingles de *General Responsibility Assignment Software Patterns*) y los Patrones Banda de los Cuatro (GOF por sus siglas en ingles de *Gang of Four*), encargados de la inicialización, agrupación y comunicación de los objetos. A continuación, se describen los patrones utilizados en el desarrollo de la propuesta en la presente investigación:

2.5.2 Patrones GRASP

- **Experto en Información:** permite asignar responsabilidades específicas a cumplir a las clases, de acuerdo con la información que maneja (Larman, 1999). (Ver figura 6).

Capítulo 2: Propuesta de solución

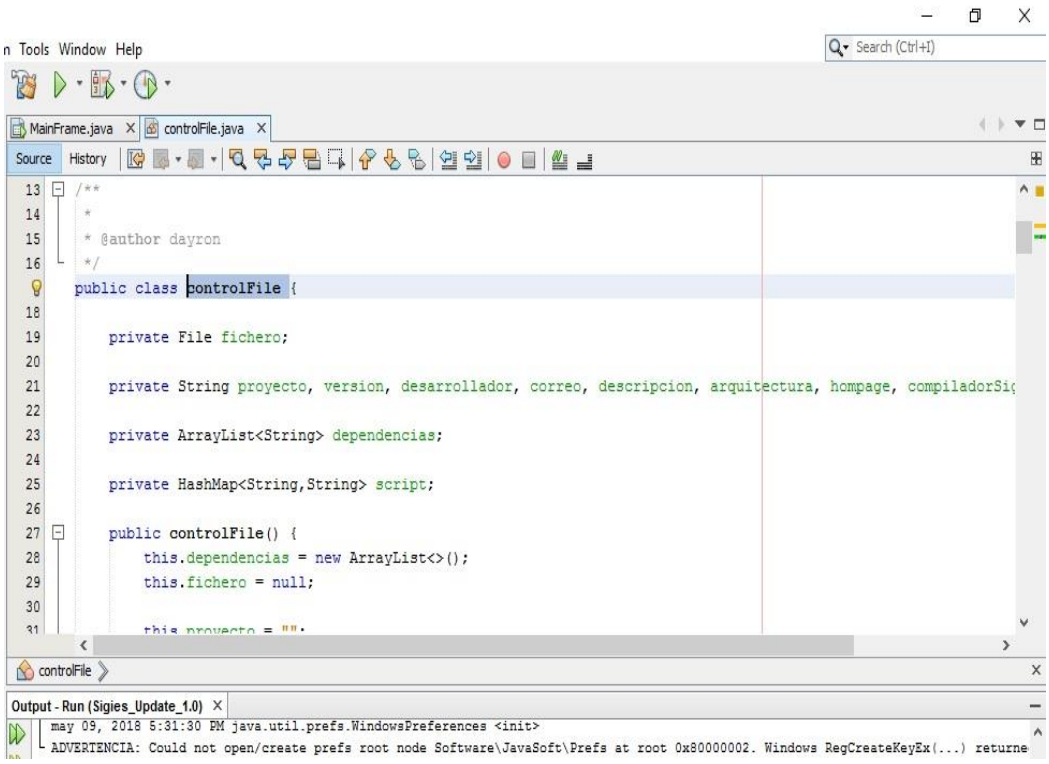


Figura 6: Experto en Información (Elaboración Propia)

- **Bajo Acoplamiento:** el acoplamiento mide la fuerza con que una clase está conectada a otra, de esta forma una clase con bajo acoplamiento debe tener un número mínimo de dependencia con otras clases (Patrones, 2016).
- **Alta cohesión:** la cohesión mide el grado en que están relacionadas las responsabilidades de una clase (Patrones, 2016).
- **Polimorfismo:** El polimorfismo es la propiedad que permite que una misma operación sea aplicada a varios objetos de distintos tipos (Patrones, 2016). (Ver figura 7).

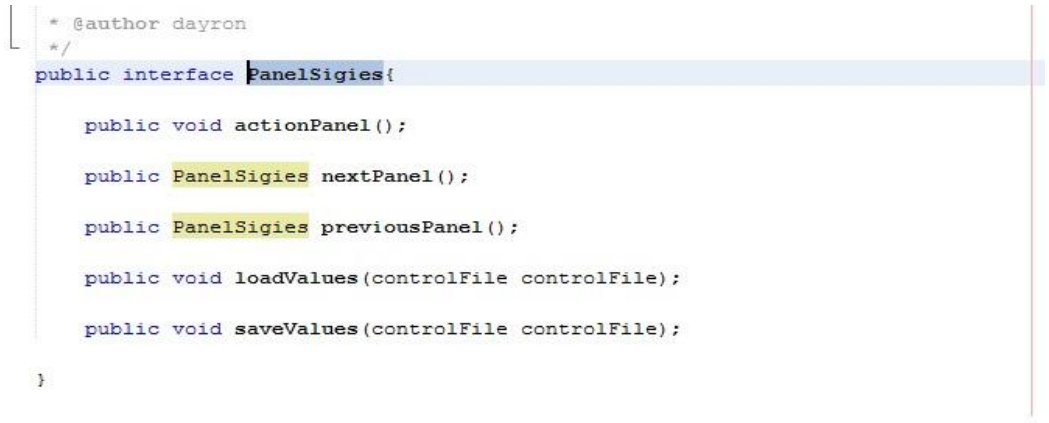


Figura 7: Patrón polimorfismo

2.5.3 Patrones GOF

- **Instancia única (*Singleton*):** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. (Rojas, 2014). (Ver figura 8).

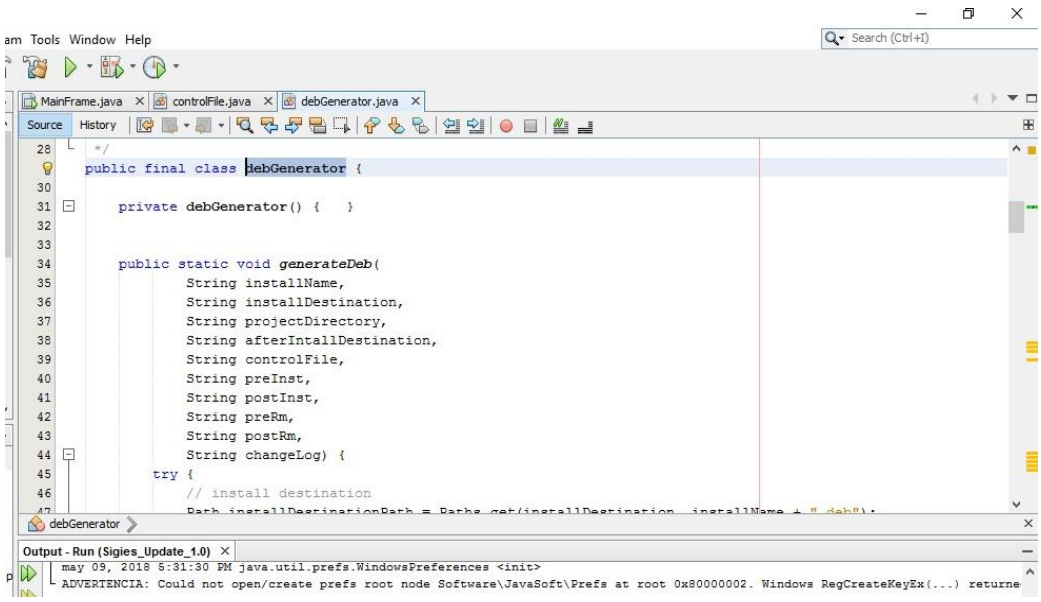


Figura 8: Singleton (Elaboración Propia)

2.6 Diagrama de clase

Para modelar clases, incluidos sus atributos, operaciones, relaciones y asociaciones con otras clases, el UML proporciona un diagrama de clase, que aporta una visión estática o de estructura de un sistema, sin mostrar la naturaleza dinámica de las comunicaciones entre los objetos de las clases (Pressman, 2010). (Ver figura 9).

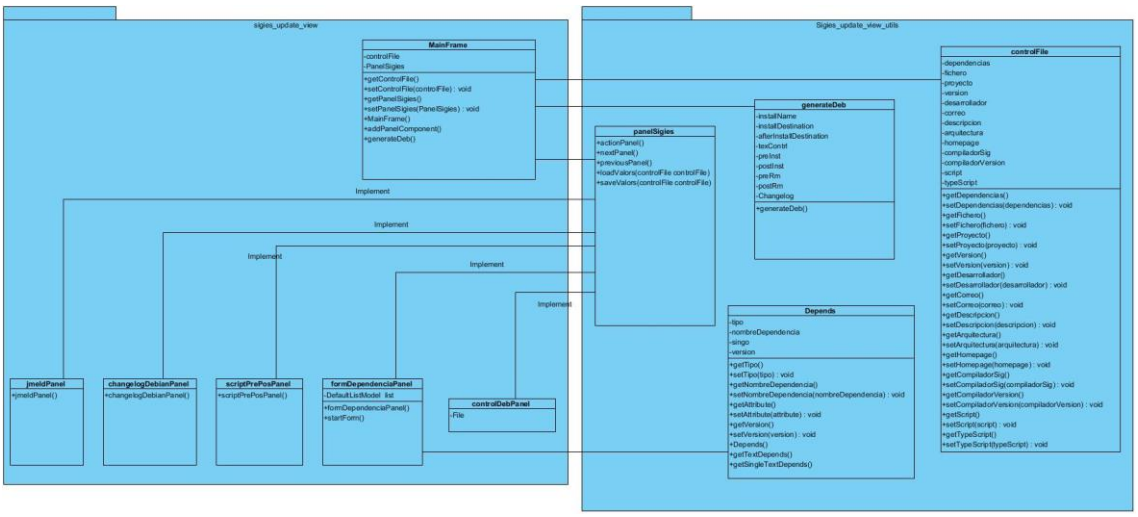


Figura 9: Diagrama de clases (Elaboración propia)

Capítulo 2: Propuesta de solución

2.7 Estándares de codificación

Un estándar de codificación comprende aspectos de la generación de código que los programadores deben implementar para lograr una herramienta con calidad. El propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme.

Usar técnicas de codificación sólidas y aplicar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para obtener un buen rendimiento y contribuir a la calidad del software (Microsoft 2017). Por lo antes expuesto, el autor de la presente investigación define el estándar de codificación: **LowerCamelCase** el cual define la letra inicial de la primera palabra en minúscula y las iniciales del resto de las palabras en mayúscula, (Ver figura 10).

Figura 10: Estándar LowerCamelCase (Elaboración propia)

Conclusiones parciales

La confección del modelo de dominio permitió obtener una representación de los conceptos que intervienen en el desarrollo de la herramienta para el empaquetamiento de actualizaciones de SIGIES.

Los requisitos funcionales y no funcionales identificados permitieron determinar las principales funcionalidades a desarrollar, con el propósito de darle cumplimiento al objetivo propuesto.

A partir de los requisitos funcionales se realizaron las historias de usuario para determinar el tiempo que demora realizar cada requisito.

Capítulo 2: Propuesta de solución

Los patrones de diseño seleccionados constituyeron una guía para la implementación de la propuesta de solución.

Con el propósito de que la herramienta cumpla con todas las funcionalidades identificadas y que fuera implementadas sin dificultad, se realizó el diagrama de clase.

Capítulo 3: Validación de la solución

Capítulo 3 Validación de la Solución

Una de las fases más importantes en el desarrollo de toda herramienta de software es la de pruebas, en esta, se trata de comprobar la calidad del producto; por tanto, si se cumplen los objetivos propuestos mediante la verificación de los requerimientos, ya sea funcional como no funcional, entonces el software es de calidad. En el presente capítulo se llevará a cabo la etapa de pruebas de software necesaria para establecer el nivel de conformidad del cliente con la herramienta.

3.1 Pruebas de Software

El desarrollo de un software es una tarea compleja y son innumerables las posibilidades de errores por lo que este ha de ir acompañado de alguna actividad que garantice la calidad; las pruebas son un elemento crítico para la garantía de calidad del software. Estas son actividades en las cuales un sistema o componente es ejecutado bajo condiciones o requerimientos especificados, los resultados son observados y registrados, además se realiza una evaluación del sistema o componente. Las pruebas y validación de los resultados se ejecutan en cada una de las etapas de desarrollo. Se realizan una y otra vez hasta que sean erradicados todos los errores que presenta el sistema y se pueda comprobar la eficiencia de las operaciones utilizadas para satisfacer las necesidades del cliente. (Pressman, 2010).

De acuerdo a la IEEE[3] el concepto de prueba se define como: *"Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente"*. De aquí la importancia que tiene realizarle pruebas al software antes de ser entregado al usuario final, garantizando así que el mismo tenga la mejor calidad posible.

Objetivo de las pruebas:

- Validar y probar los requisitos que debe cumplir el software.
- Verificar que los requisitos fueron implementados correctamente.
- Encontrar y documentar los defectos que puedan afectar la calidad del software.
- Verificar que el software trabaje como fue diseñado (Pressman, 2010).

[3] (Institute of Electrical and Electronics Engineers) en español **Instituto de Ingenieros Eléctricos y Electrónicos**

Capítulo 3: Validación de la solución

3.2 Métodos de Pruebas

Caja Blanca: La prueba de caja blanca, en ocasiones llamada *prueba de caja de vidrio*, es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que: 1) garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez, 2) revisen todas las decisiones lógicas en sus lados verdadero y falso, 3) ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas y 4) revisen estructuras de datos internas para garantizar su validez. (Pressman, 2010)

Caja Negra: Las pruebas de caja negra, se centran en los requisitos funcionales del software, que permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (Mera-Paz, 2016).

3.3.1 Niveles de Prueba

Existen 4 niveles de prueba entre ellos se encuentra, las pruebas unitarias, las pruebas de integración, las pruebas de sistemas, las pruebas de aceptación, las pruebas de humo, las de alpha (Mera-Paz, 2016). En acorde a la guía de la metodología, con el objetivo de lograr las pruebas necesarias a la herramienta desarrollada, las pruebas internas y las pruebas de aceptación según los requisitos planteados por el cliente fueron las escogidas, dejando claro que no sería necesario aplicar los otros niveles.

Pruebas Internas: En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas (Rodríguez, 2015).

El método de caja blanca, es el adecuado para una herramienta de escritorio, donde el traceo del código fuente es la base de este método de prueba. El método fue aplicado durante el período final de implementación de cada funcionalidad, donde se utilizó la técnica de ruta básica.

Ruta Básica: La prueba de ruta o trayectoria básica es una técnica de prueba de caja blanca propuesta para derivar una medida de complejidad lógica de un diseño de procedimiento y usar una medida como guía para definir un conjunto básico de rutas

de ejecución. Los casos de prueba derivados para revisar el conjunto básico tienen garantía para ejecutar todo enunciado en el programa, al menos una vez durante la prueba (Pressman, 2010).

Se decide aplicar esta técnica al método generateDeb () de la clase debGenerator () (Ver figura 11).

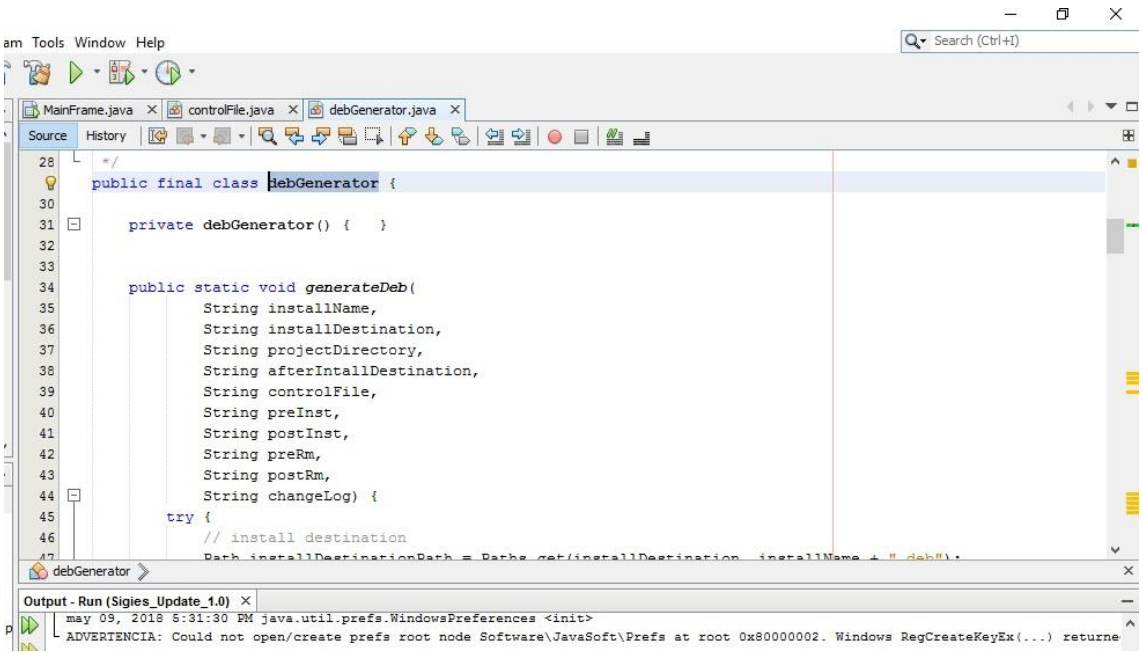


Figura 11: Método generateDeb (Elaboración propia)

El siguiente paso sería conformar el gráfico que representa la estructura lógica de los procedimientos del método seleccionado para la ruta básica. (Ver figura 12).

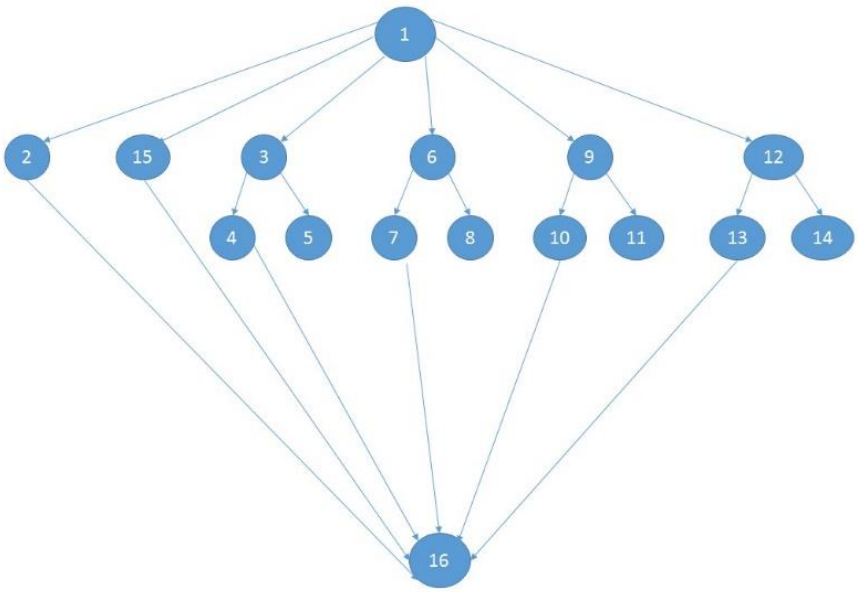


Figura 12: Grafo de Ruta Básica (Elaboración propia)

Capítulo 3: Validación de la solución

Luego se procederá a calcular la complejidad ciclomática:

V(G): Complejidad ciclomática.

E: Cantidad de Aristas.

N: Cantidad de Nodos.

P: Nodos Predicativos.

R: Regiones.

$$V(G) = E – N + 2 = 6$$

$$V(G) = P + 1 = 6$$

$$V(G) = R = 6$$

El valor calculado como V(G), define el número de caminos independientes del conjunto básico, lo que facilitó conocer el número de pruebas que se deben realizar. En el paso siguiente se representará los caminos independientes encontrados.

- 1 - 2 - 16
- 1 - 3 - 4 - 16
- 1 - 6 - 7 - 16
- 1 - 9 - 10 - 16
- 1 - 12 - 13 - 16
- 1 - 15 - 16

Luego de haber encontrado los caminos básicos, se procede a diseñar los 6 casos de pruebas por cada uno de estos caminos. A continuación, se evidencia 2 de ellos.

Para cada uno de los casos de prueba es necesario tener en cuenta: descripción, condición de ejecución, entrada y resultados esperados.

Tabla 5: Caso de prueba RB-1 (Elaboración propia)

Código: CPRB-01	
Descripción:	Crear el Archivo Control, para efectuar el empaquetamiento.

Condición de evaluación:	Debe crear el archivo control con sus datos correspondientes.
Entrada:	Datos del archivo Control.
Resultados esperados:	La confección del archivo en cuestión.
Evaluación de prueba: Satisfactoria.	

Tabla 6: Caso de Prueba RB-2 (Elaboración propia)

Código: CPRB-02	
Descripción:	La creación de los Scripts definidos por el usuario.
Condición de evaluación:	Debe Crear el archivo de configuraciones y ejecutar los comandos programados.
Entrada:	Código del Scripts
Resultados esperados:	La Confección de archivo en cuestión.
Evaluación de prueba: Satisfactoria.	

Luego de haber realizado los casos de pruebas para las 6 rutas, se obtuvo como resultado la acreditación de ser satisfactoria cada prueba realizada.

A continuación, se da paso a las pruebas de caja negra con la técnica escogida de partición equivalente, mediante descripciones de casos de pruebas.

Resultados

Para la realización de las pruebas de caja negra se empleó la técnica partición de equivalencia. Esta técnica permite examinar los valores válidos e inválidos de las entradas existentes en el software. A continuación, es sometida a verificación las HU-1 y la HU-3.

Tabla 7: Caso de prueba HU 1 (Elaboración propia)

	CP. Historias de usuario: HU-1	
Nombre: Incluir ficheros		
Descripción: La aplicación debe permitirle al usuario incluir los ficheros para poder hacer la comparación.		
Acción Probar	**Datos de entrada**	**Resultados esperados**
Cargar archivo	Fichero 1 Fichero 2	Mostrar el archivo cargado
Evaluación de las pruebas: Satisfactoria		

Tabla 8: Casos de prueba HU-3 (Elaboración propia)

	CP. Historias de usuario: HU-3	
Nombre: Comparar ficheros.		
Descripción:		
Acción Probar	**Datos de entrada**	**Resultados esperados**
	Fichero 1 Fichero 2	Mostrar los resultados encontrados en la comparación.
Evaluación de las pruebas: Satisfactoria		

Luego de realizar los casos de pruebas con el método de caja negra, se representa las 3 iteraciones realizadas. (Ver figura 13).

Capítulo 3: Validación de la solución

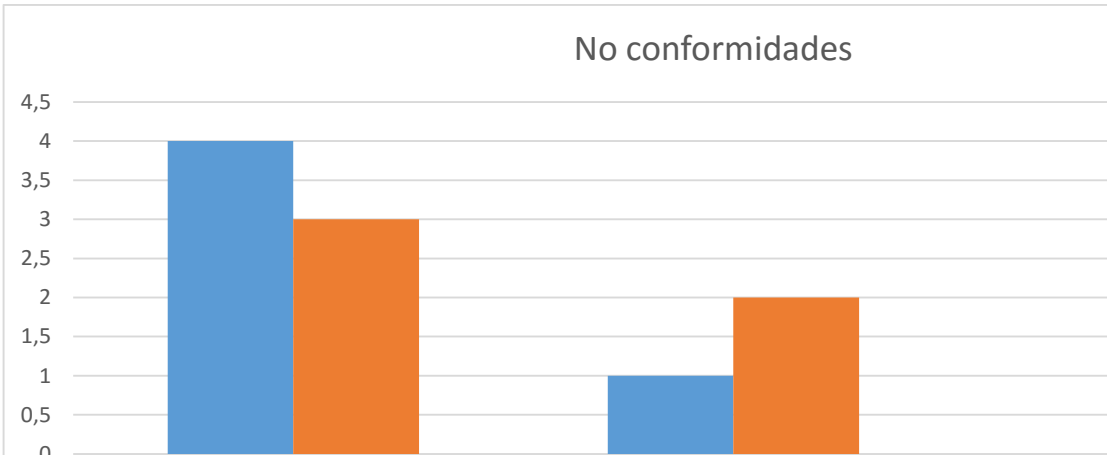


Figura 13: No conformidades (Elaboración propia)

Una vez realizadas las pruebas de caja negra, se obtuvo como resultado que al terminar la tercera iteración se erradicaron las no conformidades. Estas pruebas permitieron mejorar la manera de interactuar el usuario con la herramienta y se erradicaron los errores funcionales evidenciados en las dos primeras iteraciones.

3.3 Pruebas de Aceptación

Son las pruebas finales antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales fue construido. (Rodríguez, 2015). Para la realización de esta tarea se define utilizar el proceso llamado prueba alfa; el software se utiliza en un escenario natural con el desarrollador registrando los errores y problemas de uso detectados por los clientes (Pressman, 2010).

Para el comienzo de las pruebas se estableció las siguientes definiciones con el objetivo de clasificar las no conformidades del cliente:

- Complejidad alta, perteneciente a las funcionalidades de la herramienta.
- Complejidad media, pertenecientes a las conformidades en las interfaces de la herramienta.
- Complejidad baja, para indicar errores ortográficos y en los casos de prueba.

Para comenzar estas pruebas fueron establecidas 3 iteraciones, en la primera se evidenciaron 4 no conformidades de clasificación media y 2 de clasificación baja, luego de haber terminado con estas revisiones, se decide aplicar pruebas de regresión para comprobar que las no conformidades detectadas hasta el momento, fueron resueltas. Continuando con la segunda iteración se registró 4 no conformidades de clasificación

alta, 2 de media y una sola de baja. Luego de haber resuelto estas no conformidades se decide aplicar la tercera iteración, arrojando como resultados que todas las no conformidades detectadas, fueron solucionadas.

A continuación, se grafican los resultados de las pruebas de aceptación durante las 3 iteraciones realizadas. (Ver figura 14).

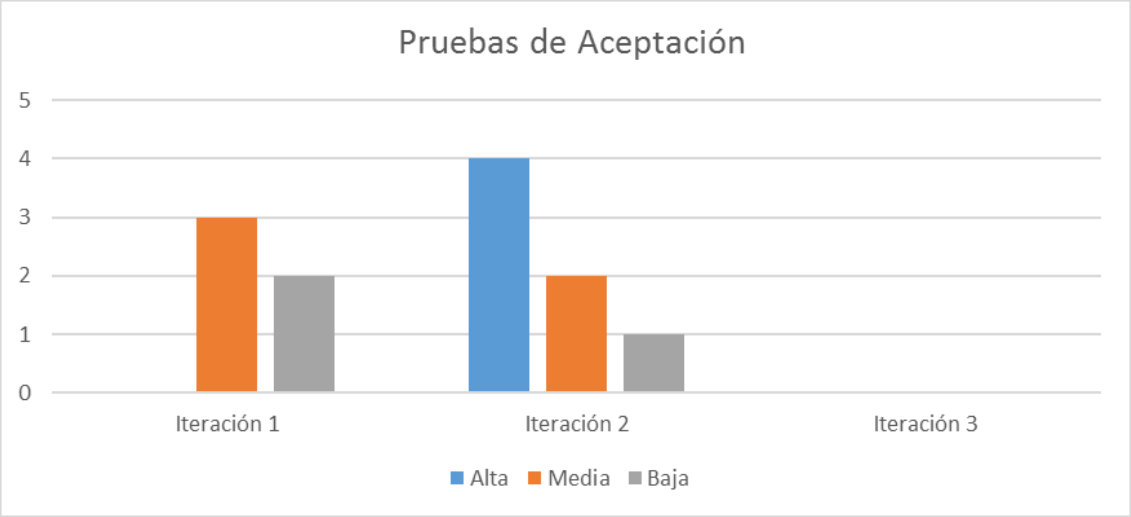


Figura 14: No conformidades (Elaboración propia)

Para facilitar este proceso de prueba, indicado por la metodología fue necesario diseñar los casos de pruebas correspondientes a cada requisito funcional. Donde a continuación se evidencia dos de ellos. (Ver figura 15 – 16).

Condiciones de ejecución
Se debe haber realizado la comparación de directorios o ficheros.
Se debe haber realizado las configuraciones del paquete Debian.

SC 1 Ejecutar el empaquetamiento

Escenario	Descripción	Respuesta del sistema	Flujo central
EC **1.1** Ejecutar el empaquetamiento.	El usuario selecciona la opción Package.	El sistema permite empaquetar el fichero o directorio seleccionado una vez validadas las dependencias y muestra un mensaje de información.	Package/ New/ New/ Directory Comparison/ Browse/ Next/ Next/ Next/Package Package/ New/ New/ File Comparison/ Browse/ Next/ Next/ Next/Package

Figura 15: Caso de Prueba Requisito Funcional 7 (Elaboración propia)

Capítulo 3: Validación de la solución

SC 1 Comparar directorios

Escenario	Descripción	Respuesta del sistema	Flujo central
EC **1.1** Realizar comparación.	El usuario selecciona los directorios y seguidamente la opción Aceptar.	El sistema muestra los resultados de comparar dos directorios seleccionados. Se visualizan los datos: -Nombre del directorio -Dirección de directorio -Contenido del directorio -L File (representa los ficheros del directorio Left) -Size (tamaño de los ficheros de los directorios) -R File (representa los ficheros del directorio Right) -L y R (Cambios en los ficheros Left y Right respectivamente) -Sep (columna que permite separar un directorio de otro) Y además se muestran las opciones: -Desplegar directorio -Plegar directorio -Vista de los directorios -Abrir último fichero abierto -Volver a comparar los directorios -Filtros -Modificar el contenido de los directorios -Next	Package/ New/ New/ Directory Comparison/ Browse/ Aceptar
EC **1.2** Opción Desplegar directorio.	El usuario selecciona el ícono más.	El sistema permite desplegar los directorios comparados.	Package/ New/ New/ Directory Comparison/ Browse/ Aceptar/ Ícono más (+)

Figura 16: Caso de Prueba Requisito Funcional 9 (Elaboración propia)

Estos diseños de casos de pruebas permitieron comprobar que la herramienta desarrollada proporcionaba soluciones correctas para cada requisito.

Resultados

Con las pruebas de aceptación, se comprobó que el cliente está satisfecho con la solución propuesta la cual posibilita minimizar las deficiencias a la hora de realizar las actualizaciones y el empaquetado de la nueva versión de SIGIES. El **Anexo 5** hace referencia a la carta de aceptación firmada por el cliente donde especifica el nivel de aceptación obtenido por la HEAS-SIGIES.

Conclusiones parciales

Luego de haber realizado las pruebas de caja blanca, para la validación de la propuesta de solución, se comprobó que la herramienta persigue una buena calidad y eficiencia en sus procesos.

Las pruebas de caja negra, permitieron obtener resultados satisfactorios después de haber culminado las 3 iteraciones, debido a que se eliminaron los errores de funcionalidades de la herramienta.

Además, las pruebas de aceptación posibilitaron comprobar que el cliente estaba satisfecho con la solución propuesta.

Conclusiones

Conclusiones

- La elaboración del marco teórico de la investigación, permitió identificar los conceptos relacionados con el objeto de estudio.
- El análisis de las soluciones similares posibilitó determinar que la herramienta a desarrollar, debe integrar tanto las funcionalidades de empaquetamiento como las de comparaciones de directorios.
- Para darle cumplimiento al objetivo general, se desarrolló una aplicación haciendo uso de las tecnologías y herramientas: Visual Paradigm, NetBeans, GitLab y el lenguaje de programación Java.
- La herramienta posibilitó minimizar las deficiencias a la hora de realizar el proceso de actualización del Sistema SIGIES.
- La realización de pruebas de software mediante los métodos de caja blanca y caja negra, permitieron determinar y erradicar las no conformidades encontradas durante las 3 iteraciones realizadas.
- Las pruebas de aceptación del cliente, posibilitaron obtener la satisfacción y la conformidad de la herramienta para el proceso de empaquetamiento de actualizaciones del Sistema de Gestión para el Ingreso a la Educación Superior.

Recomendaciones

Recomendaciones

A partir del trabajo realizado se recomienda al proyecto:

- Agregar a la herramienta nuevas funcionalidades, como una base de datos que posibiliten el almacenamiento de varios archivos de actualización.

Bibliografía

Bibliografía

1. **Ambler, Scott W. 2017.** Agile Modeling. *Agile Modeling.* [En línea] 4 de enero de 2017. http://www.agilemodeling.com/essays/simpleTools.htm#SelectingCASE.
2. **AntumDeluge. 2018.** Debreate. [En línea] 2018. https://antumdeluge.github.io/debreate-web/.
3. **Aoki, Osamu. 2013.** *Guía de referencia de Debian.* 2013.
4. **Aristizábal Mejía, Nicolás y Torres Moreno, Miguel Eduardo. 2017.** Técnicas de Levantamiento. [En línea] Martes de mayor de 2017. https://www.researchgate.net/publication/244724135_Tecnicas_de_Levantamiento_de_Requerimientos_con_Innovacion.
5. **Beck, Kent . 2002.** *Extreme programming explained: embrace change.* s.l.: Addison-wesley professional, 2002.
6. **Belloch Ortí, Consuelo. 2014.** TIC. [En línea] 2014.
7. **Culturacion.com. 2017.** Culturacion.com. [En línea] 2017. http://culturacion.com/que-son-los-scripts/.
8. **Debian. 2018.** *Debian.* [En línea] 2018. https://manpages.debian.org/jessie/dh-make/dh_make.8.en.html.
9. **Debian. 2016.** Debian. *Debian.* [En línea] 5 de julio de 2016. https://www.debian.org/distrib/packages.
10. **Expósito, C. M. 2006.** *Interfaz Gráfica de usuario. Aproximación semiótica y cognitiva.* 2006.
11. **Git. 2017.** Git. [En línea] 2017. https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones.
12. **Ingram, David. 2017.** *Qué es un sistema de gestión de la información.* s.l.: Radoslav Lazarov, 2017.
13. **Johansen, Ernst. 2006.** *Patrones de diseño.* 2006. págs. 62-65.
14. **Joyanes, L. 2013.** *Fundamentos de programación. Algoritmos, estructuras de datos y objetos.* s.l.: McGrawHill, 2013.
15. **Larman, Craig. 1999.** *UML y Patrones. Pearson.* 1999.
16. **Martínez, L. H. 2008.** *Intérprete y Entorno de Desarrollo para el Aprendizaje de Lenguajes de Programación Estructurada.* 2008.
17. **Meld. 2018.** GitHub - albfan/jmeld. *jMeld.* [En línea] 2018. https://github.com/albfan/jmeld.
18. **MeldMerge. 2016.** MeldMerge. [En línea] 2016. [Citado el: 12 de diciembre de 2017.] http://meldmerge.org/.
19. **Mera-Paz, Julian Andrez. 2016.** *Análisis del proceso de pruebas.* 2016.

Bibliografía

20. **Netbeans.org. 2016.** NetBeans. *NetBeans.* [En línea] 2016. http://netbeans.org/index_es.html.

21. **Patrones. 2016.** Patrones. [En línea] 2016. http://ima.udg.edu/~sellares/EINF-ES2/Present1011/Patrons%20GRASP.pdf.

22. **Pressman, Roger S. 2010.** *Ingeniería de Software. Un enfoque práctico. Séptima Edición.* s.l.: MC Graw Hill, 2010.

23. **Real Academia Española. 2017.** Real Academia Española. *Real Academia Española.* [En línea] 2017. http://www.rae.es.

24. **Rodríguez, Tamara. 2015.** *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana: s.n., 2015.

25. **Rojas, MC Juan Carlos Olivares. 2014.** *Patrones de Diseño.* 2014.

26. **Significados.com. 2013.** [En línea] 2013. [Citado el: 11 de Diciembre de 2017.] https://www.significados.com/servidor/.

27. **Soulary, Beyris y Liliam, Celia. 2010.** *Proceso de medición y análisis para el polo de hardware y automática.* 2010. Vol. 3.

28. **Vendemia. 2014.** CONCEPTODEFINICION.DE. [En línea] 2014. http://conceptodefinicion.de/codigo-fuente/.

29. **Visconti, Marcello y Astudillo, Hernán. 2013.** *Fundamentos de Ingeniería de Software.* s.l.: Universidad Técnica Federico Santa María, 2013.

30. **Visual Paradigm. 2017.** Visual Paradigm. *Visual Paradigm.* [En línea] 23 de enero de 2017. [Citado el: 25 de octubre de 2016.] https://www.visual-paradigm.com/.

31. **WinMerge. 2018.** WinMerge. [En línea] 2018. http://winmerge.org/?lang=es.

Anexos

Anexo 1

Guía de preguntas utilizadas en el desarrollo de la entrevista aplicada a los desarrolladores a cargo de realizar la actualización de SIGIES.

1. ¿Usted conoce de soluciones informáticas que cumplan las características deseadas para la herramienta que se pretende desarrollar?
2. Teniendo en cuenta que se desea desarrollar una herramienta informática para perfeccionar el proceso de empaquetamiento de las actualizaciones de SIGIES. ¿Qué elementos considera usted necesarios que debe cumplir las funcionalidades en esta herramienta?
3. ¿El menú que presente esta herramienta debe ser de forma estática o debe desplazarse para ocupar menos espacio en pantalla?
4. ¿Para qué sistemas operativos debe ser compatible la herramienta que se desea desarrollar?
5. Mencione las especificaciones de hardware mínimas sobre las que la herramienta deba funcionar.

Anexo 2

Tabla 9: Acuerdos tomados en la tormenta de ideas (Elaboración propia)

Nº	Acuerdo	Responsable	Fecha
1	La herramienta debe tener funcionalidades que permitan incluir dos ficheros de forma sencilla.	Dayron Sagarra Barreras	Enero - 2018
2	Debe ser capaz de comparar ambos archivos y mostrar esa comparación.	Dayron Sagarra Barreras	Enero - 2018
3	Debe de poseer funcionalidades que permitan incluir los cambios de un fichero a otro y viceversa, así como mostrar y guardar estos cambios.	Dayron Sagarra Barreras	Enero - 2018
4	Debe permitir ejecutar los comandos de configuración necesarios para el proceso de la comparación de archivos y de la generación del paquete Debian.	Dayron Sagarra Barreras	Enero - 2018
5	Debe dar la posibilidad de insertar los paquetes de dependencia y realizar el empaquetamiento del directorio resultante.	Dayron Sagarra Barreras	Enero - 2018

Anexo 3

Tabla 10: Acuerdos tomados en las revisiones formales de los requisitos (Elaboración propia)

Nº	Acuerdo	Responsable	Cumplimiento
A1	Mejorar la descripción del requisito funcional Comparar Ficheros	Dayron Sagarra Barreras	Enero - 2018
A2	Mejorar la especificación del requisito funcional Ejecutar empaquetamiento	Dayron Sagarra Barreras	Enero - 2018
A3	Mejorar la especificación del requisito funcional Insertar dependencias	Dayron Sagarra Barreras	Enero - 2018
A4	Falta requisitos no funcionales relacionado con los criterios de Usabilidad, Interfaz y apariencia de la herramienta	Dayron Sagarra Barreras	Enero - 2018

Anexo 4

Tabla 11: Historias de usuario #2 (Elaboración propia)

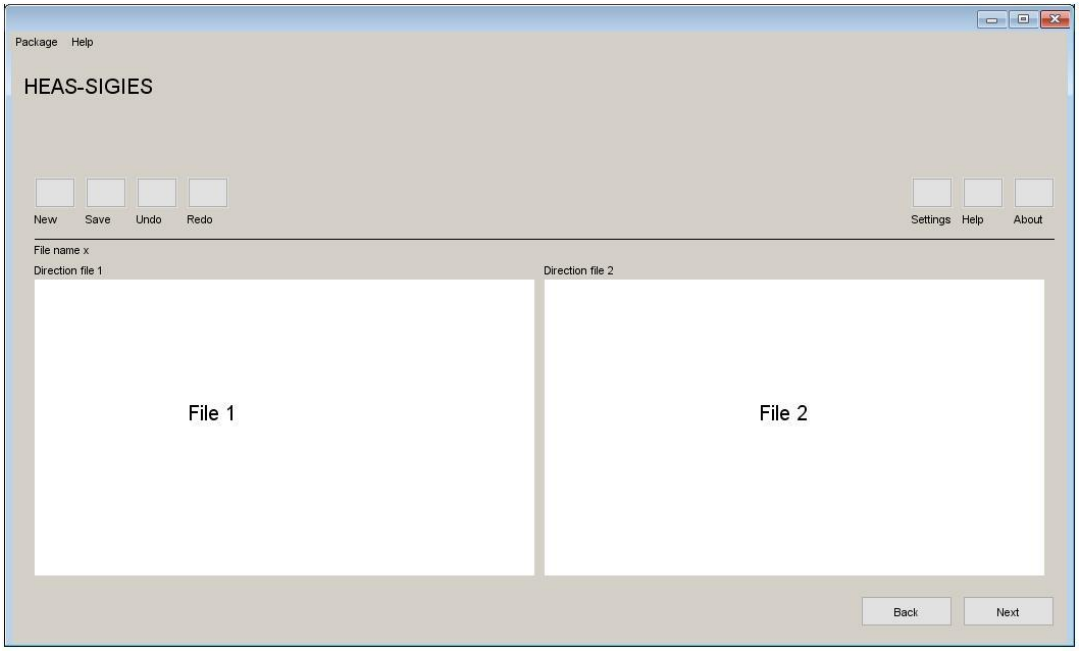
Número: 2	**Nombre del requisito:** Comparar ficheros
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 2 días
Descripción: La aplicación debe garantizar la comparación de los dos ficheros antes incluidos. El usuario selecciona los ficheros donde la herramienta muestra los resultados de comparar dos ficheros seleccionados. Se visualizan los datos: -Nombre del fichero -Dirección de fichero -Contenido del fichero -Cantidad de líneas -Cantidad de columnas	
Observaciones: antes de realizar esta operación, ya los dos ficheros a comprobar deben estar incluidos en la herramienta	
Prototipo de interfaz: 	

Tabla 12: Historias de usuario #3 (Elaboración propia)

Número: 3	**Nombre del requisito:** Modificar los datos de los ficheros.
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 4 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 5 días

Descripción: La aplicación debe ser capaz de mostrar los resultados de la comparación realizada.

- El usuario modifica los datos de los ficheros.
- La herramienta permite insertar los cambios desde un fichero hacia el otro.
- Permite guardar los cambios de los ficheros modificados.
- Permite volver atrás los cambios realizados.

Observaciones:

Prototipo de interfaz:

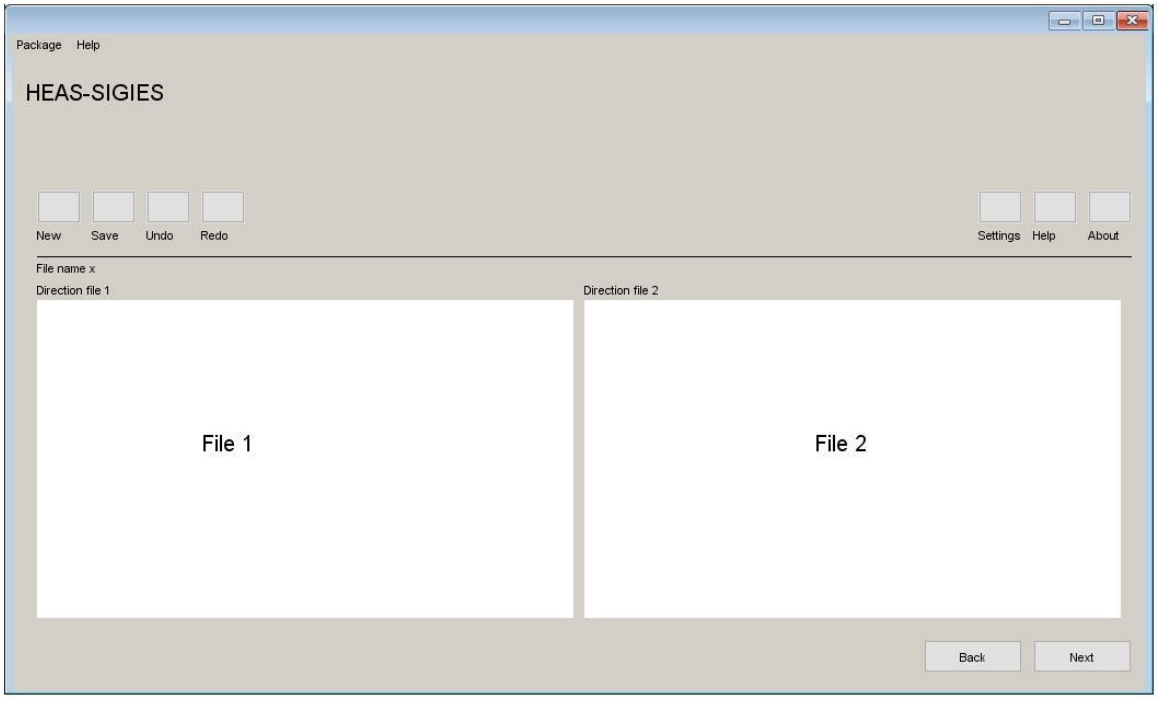


Tabla 13: Historias de usuario #4 (Elaboración propia)

Número: 4	**Nombre del requisito:** Configuraciones del paquete debian
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo Estimado:** 1 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 2 días
Descripción: Debe brindar la posibilidad de introducir las configuraciones del paquete Debian. La herramienta permite introducir los siguientes datos para llenar el formulario: -Name Project - Name Developer -Version -Email Developer -Decription -Architecture -Distribution SO -Homepage -Compilador-debhelper-Version Además, permite seleccionar las direcciones donde se va a guardar e instalar el paquete Debian: -Save .deb in -Install .deb in	
Observaciones:	
Prototipo de interfaz:	

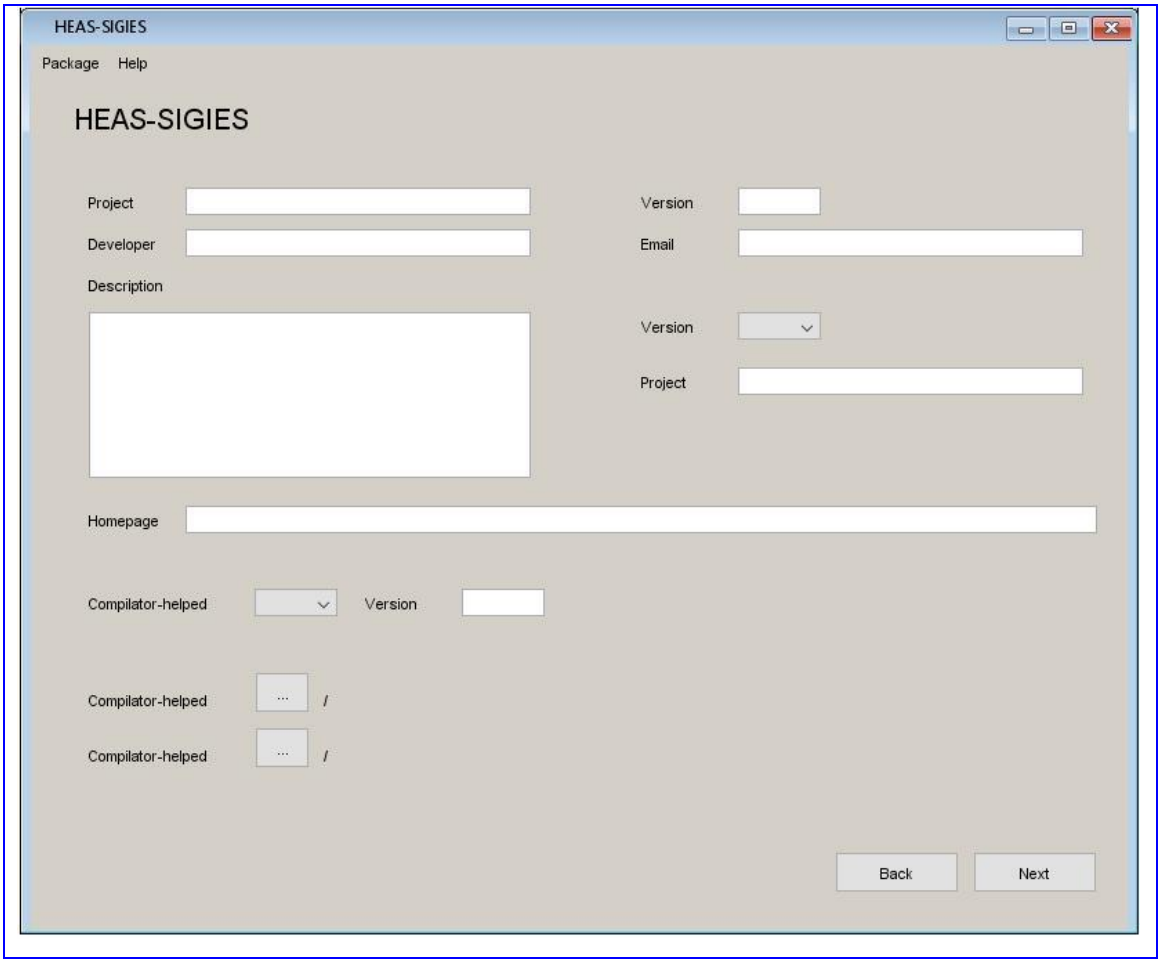


Tabla 14: Historias de usuario #5 (Elaboración propia)

Número: 5	**Nombre del requisito:** Insertar paquetes de dependencias
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 2 días
Descripción: La aplicación debe permitir incluir dependencias. La herramienta debe permitir incluir las dependencias y para ello debe introducir los siguientes campos: -Depends -Version -Type Depends	

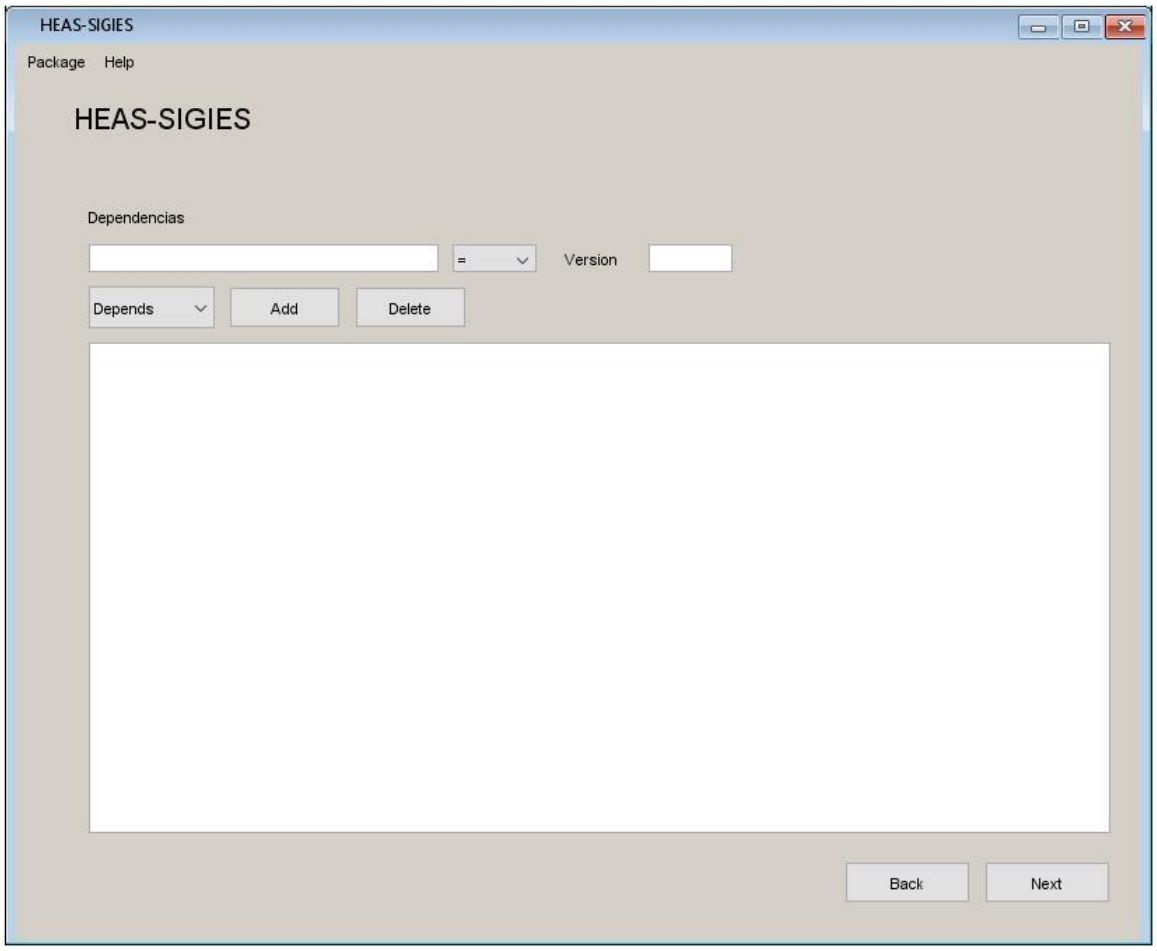
Observaciones: un paquete de dependencia, no es más que las aplicaciones, incluyendo framework, lenguajes y configuraciones necesarias para que el ejecutable .deb funcione correctamente.
Prototipo de interfaz: 

Tabla 15: Historias de usuario #6 (Elaboración propia)

Número: 6	**Nombre del requisito:** Ejecutar comandos de configuración.
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 8 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 10 días
Descripción: La aplicación debe permitir incluir configuraciones pre-pos install y remove. La herramienta debe permite insertar uno o varios comandos para crear un archivo de configuración, debe seleccionar e introducir los siguientes datos:	

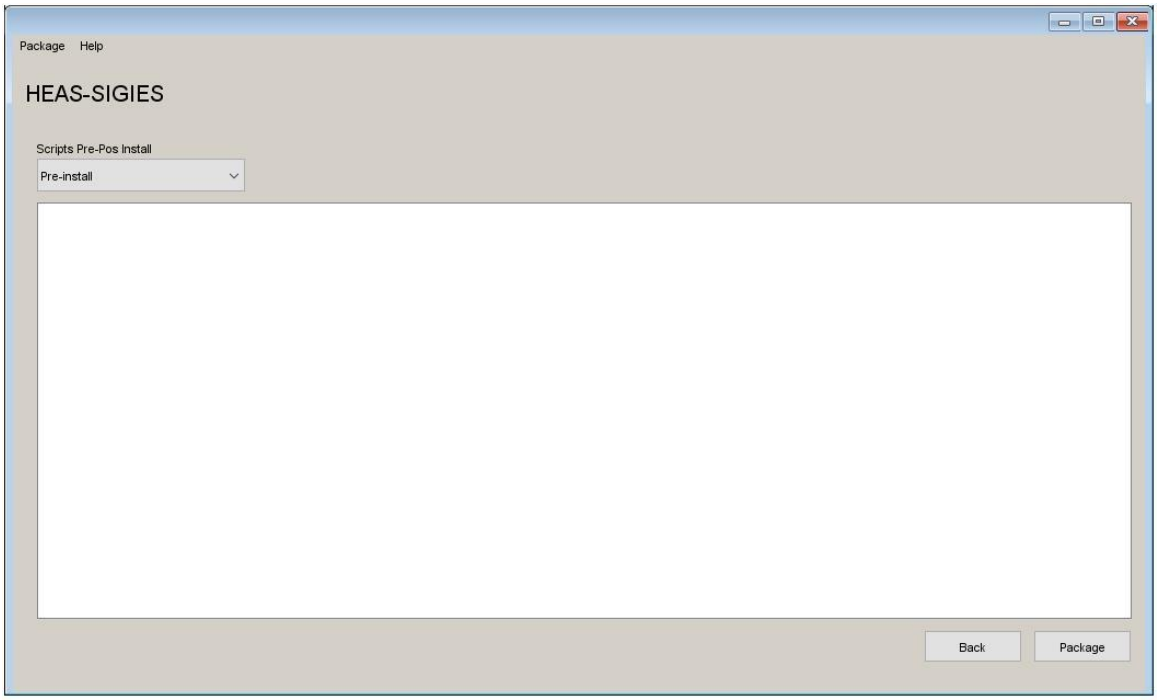
-Scripts Pre-Pos -Scripts (Entrada)
Observaciones: los Archivos Scripts, son parte de la estructura de un paquete .deb, son funcionalidades que se ejecutan antes y después de la instalación del paquete y la eliminación del mismo.
Prototipo de interfaz: 

Tabla 16: Historias de usuario #7 (Elaboración propia)

Número: 7	**Nombre del requisito:** Ejecutar el empaquetamiento.
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 4 días
Descripción: La aplicación debe permitir realizar el empaquetamiento del directorio actualizado.	
Observaciones:	

Prototipo de interfaz:

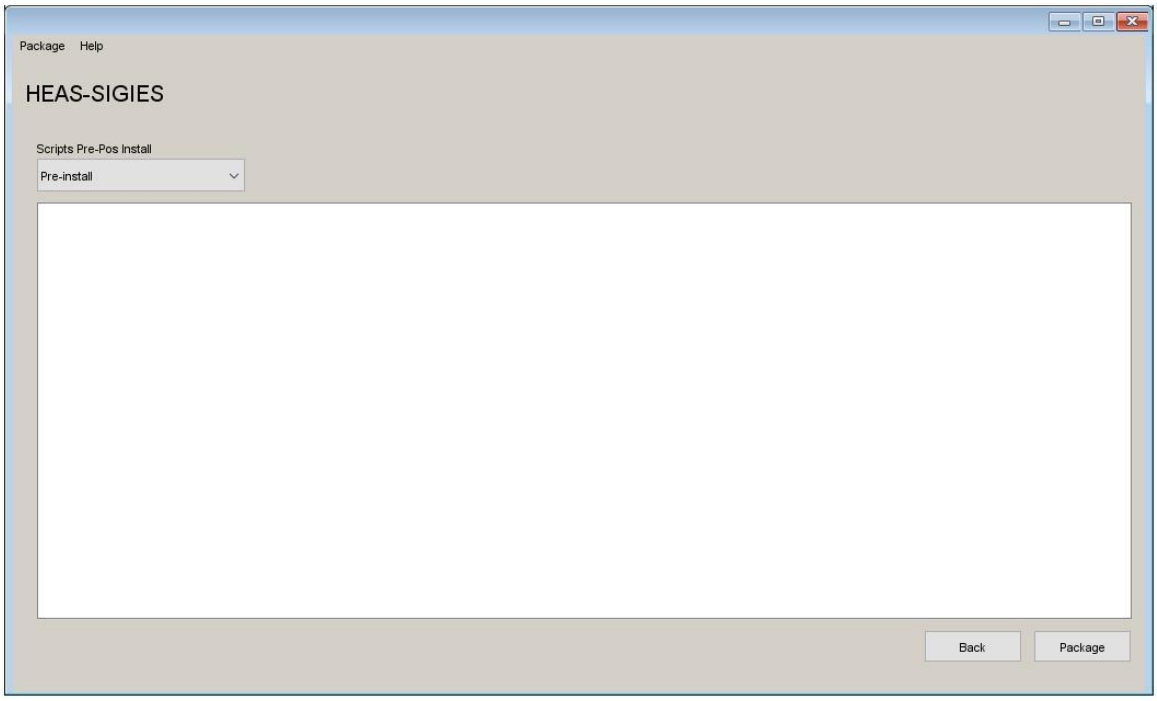


Tabla 17: Historias de usuario #8 (Elaboración propia)

Número: 8	**Nombre del requisito:** Incluir directorios
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 2 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 2 días
Descripción: La aplicación debe permitirle al usuario incluir directorios para poder realizar la comparación. 1. El usuario debe seleccionar la opción New que se encuentra en el menú Package. 2. El usuario selecciona la otra opción New que implicaría buscar directorios	
Observaciones: Debe permitir levantar el explorador de archivos para la búsqueda del directorios a incluir	
Prototipo de interfaz:	

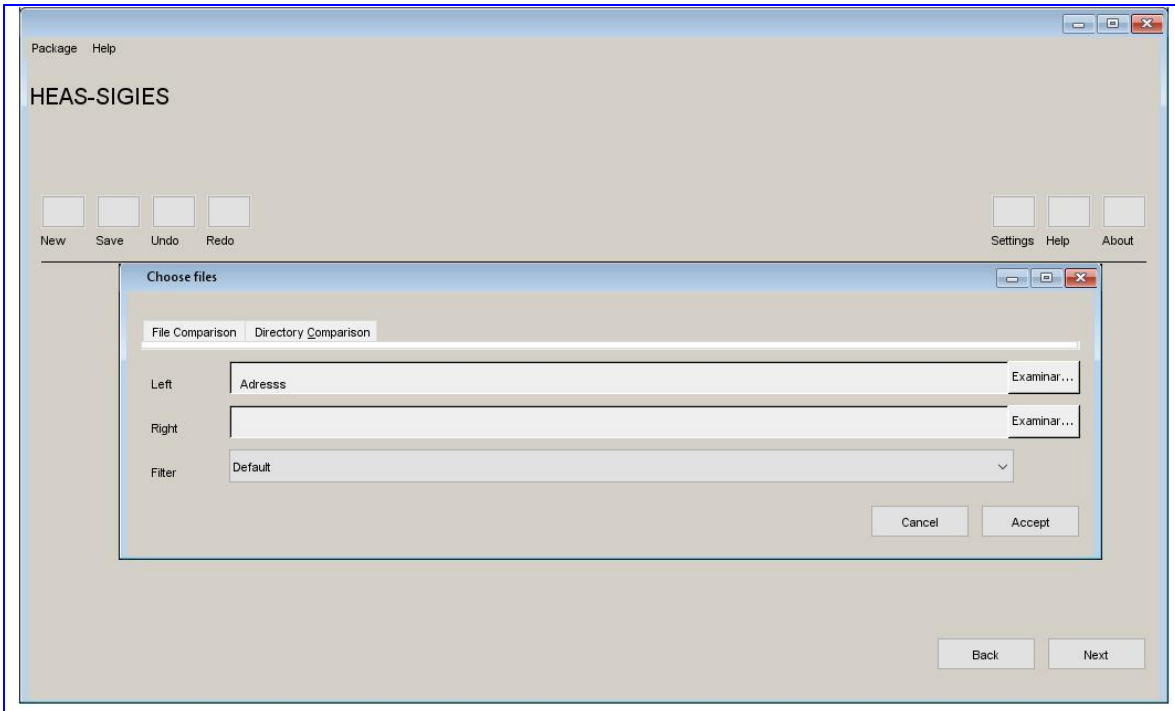


Tabla 18: Historias de usuario #9 (Elaboración propia)

Número: 9	**Nombre del requisito:** Comparar directorios.
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 4
Prioridad: Alta	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 2 días

Descripción: La aplicación debe ser capaz de comparar los Directorios seleccionados

- El usuario selecciona los directorios y seguidamente la opción Accept.
- El sistema muestra los resultados de comparar dos directorios seleccionados. Se visualizan los datos:

  -Nombre del directorio

  -Dirección de directorio

  -Contenido del directorio

  -L File (representa los ficheros del directorio Left)

  -Size (tamaño de los ficheros de los directorios)

  -R File (representa los ficheros del directorio Right)

  -L y R (Cambios en los ficheros Left y Right respectivamente)

  -Sep (columna que permite separar un directorio de otro)

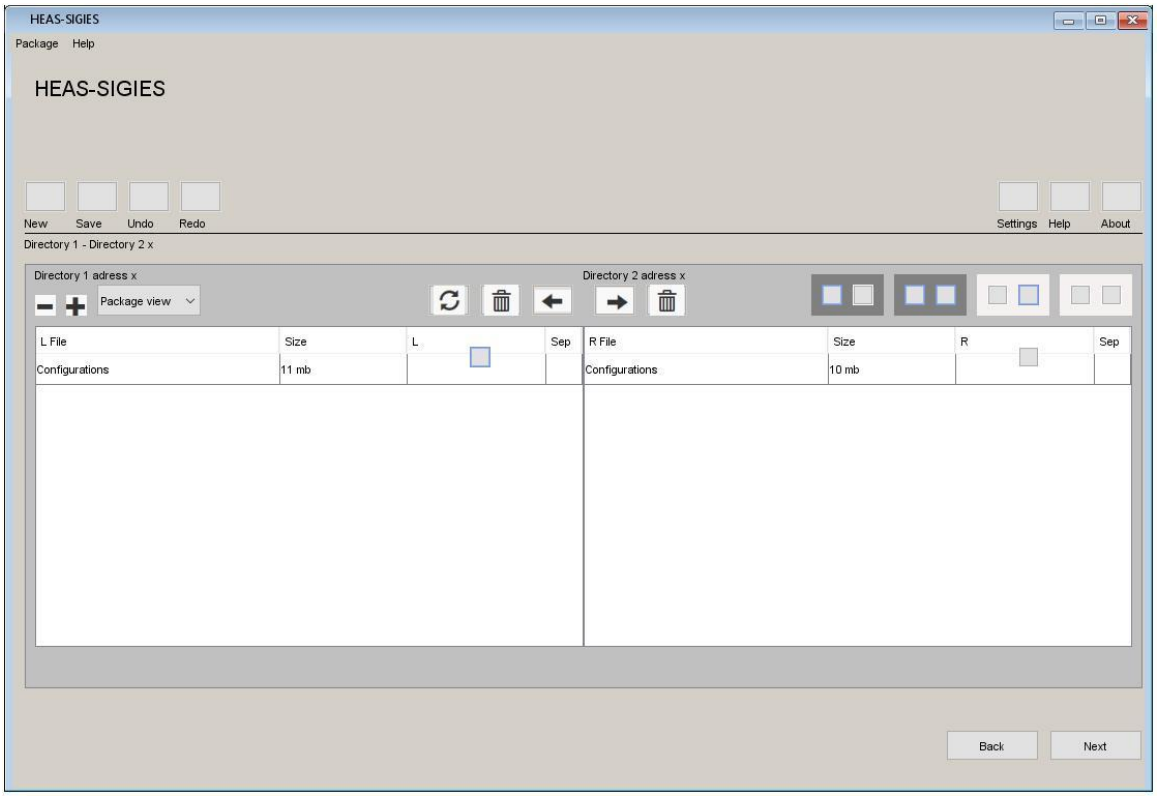
Observaciones: Deben existir dos directorios a comparar.
Prototipo de interfaz: 

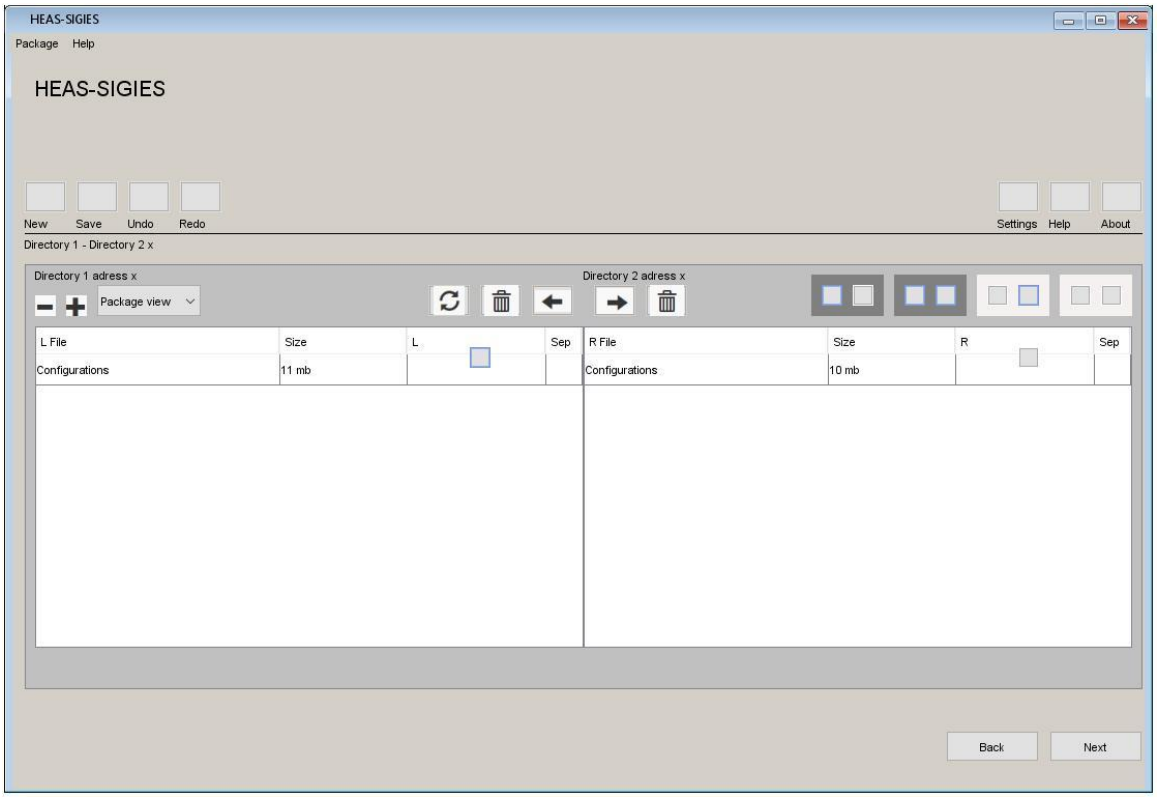
Tabla 19: Historias de usuario #10 (Elaboración propia)

Número: 10	**Nombre del requisito:** Mostrar resultados de la comparación de directorios.
Programador: Dayron Sagarra Barreras	**Iteración Asignada:** 4
Prioridad: Alta	**Tiempo Estimado:** 4 días
Riesgo en Desarrollo: Alto	**Tiempo Real:** 5 días
Descripción: Debe brindar la posibilidad de mostrar el resultado de la comparación de directorios La herramienta debe modificar los directorios e insertar el contenido de un directorio en otro. Además, muestra las opciones de: -Eliminar -Mover	

-Modificar datos de un fichero

Observaciones:

Prototipo de interfaz:



Anexo 5

CARTA DE ACEPTACIÓN DEL CLIENTE

Tesis: Herramienta para el proceso de empaquetamiento de actualizaciones en el Sistema de Gestión para el Ingreso a la Educación Superior

Descripción: En la presente carta se hará constancia de que los requisitos funcionales planteados, llegaron a ser resueltos correctamente en la herramienta mencionada con anterioridad, así confirmando el acuerdo entre el cliente y el desarrollador de la herramienta en cuestión, que se resolvió el problema planteado. Y Aclarando la calidad y la eficiencia con la que se pretendía trabajar.

Observaciones:

Para que así conste firma la presente a los ___ días del mes de _____ del año _____.

_____

Jefe de Proyecto

Ing. Jorge Luis Piña González.

Figura 17: Carta de aceptación del cliente