UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

CENTRO DE SOFTWARE LIBRE

HERRAMIENTA DE COMPILACIÓN DISTRIBUIDA DE NOVA

Trabajo de diploma presentado en opción al título de Ingeniero en Ciencias Informáticas

Autor

Humberto Pérez Arbolaez

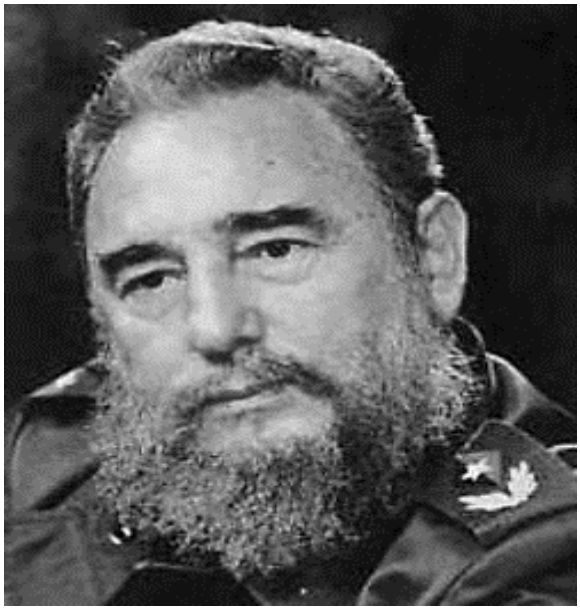
Tutores

Ing.Yaiselis Ramírez Mastrapa

Ing.Luis Daniel Sierra Corredera

LA HABANA, Junio 2018

AÑO 60 DE LA REVOLUCIÓN



"Pero batalla de ideas no significa solo principios, teoría, conocimientos, cultura, argumentos, réplica y contrarréplica, destruir mentiras y sembrar verdades; significa hechos y realizaciones concretas. Aun en período especial, bajo el bloqueo, la hostilidad y las amenazas del imperio más poderoso que ha existido, nuestro pueblo diseña y construye la más justa y humana sociedad que hasta hoy se ha conocido. A la vanguardia de esa grandiosa obra, están los jóvenes, los estudiantes y nuestros maravillosos niños. Por ello es más fuerte cada día nuestro optimismo y confianza en el porvenir."

Comandante Fidel Castro Ruz

AGRADECIMIENTOS

Mi felicidad no estaría completa el día de hoy si me llego a olvidar de esas personas que estuvieron presentes para ayudarme, por eso sería imperdonable dejar de agradecer a todos ellos. Agradezco:

A mi madre por traerme a este mundo, por sus sacrificios, por ser la luz de mis ojos y la voz de mi conciencia. Por siempre apoyarme en las buenas y en las malas decisiones que he tomado en la vida. Por la preocupación durante todo este tiempo y por siempre estar orgullosa de su hijo.

A mi padre por alentarme a continuar creciendo en la vida, por su apoyo y consejos en todo momento.

A mis hermanos por hacerme compañía en este mundo, brindarme su apoyo y hacer más completa la vida.

A mis tutores por el apoyo y la paciencia brindada durante todo este proceso.

A Ivaniet Díaz Romeu por el apoyo y la paciencia brindada durante todo este proceso.

Al tribunal por la ayuda y orientación en este último mes.

A mis profesores de todos estos años por ayudar en mi formación como profesional.

A mis compañeros de fiesta Javier Piñeiro Cárdenas, Elvis Salabarria Aquino, Alejandro rodas cueto, José Ernesto Cortes Mendez, Nestor Llerena rivera por la ayuda brindada y por aguantarme todos estos años.

A mis prietas queridas como las llamo cariñosamente Aideé Mora Urdaneta y Dareyna de la C Muñoz González por su amistad, apoyo y por aguatarme todos estos años.

A todos mis amigos y familiares que de alguna forma contribuyeron a materializar este sueño, en especial a mi grupo 1503, nunca me olvidaré de ustedes.

En general a todas las personas que de una forma u otra hicieron posible cumplir mi sueño de ser ingeniero informático. A todos muchas gracias.

DEDICATORIA

A mi familia por todo su apoyo y comprensión en los momentos más difíciles durante la universidad, especialmente a mi mamá, mi papá y hermanos que son lo más grande que tengo en esta vida.

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo Humberto Pérez Arbolaez, con carné de identidad 93021711225 soy el autor principal del trabajo titulado Herramienta de compilación distribuida de Nova y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firman la presente a los __ días del mes de junio del año 2018.

_____

Humberto Pérez Arbolaez

Autor

_____ _____

Ing**.** Yaiselis Ramírez Mastrapa

Tutora

Ing. Luis Daniel Sierra Corredera

Tutor

DATOS DE CONTACTO

Ing**.** Yaiselis Ramírez Mastrapa
Universidad de las Ciencias Informáticas, La Habana, Cuba
Correo: yaiselis@uci.cu

Ing. Luis Daniel Sierra Corredera
Universidad de las Ciencias Informáticas, La Habana, Cuba
Correo: ldsierra@uci.cu

RESUMEN

Compilar un programa en un sistema GNU/Linux es una tarea para usuarios con experiencia en este arte, dado que cada programa tiene sus particularidades, así como cada computadora o arquitectura necesita de una compilación específica para su tipo. En Cuba existe una gran variedad de computadoras con tecnologías antiguas y que dada su necesidad aún están en servicio, por tanto, es común tener que realizar la compilación de paquetes para cada uno de estos tipos de ordenadores. En el proyecto Nova, donde se desarrolla la distribución cubana GNU/Linux Nova, que está a la vanguardia en el proceso de migración nacional, se necesita compilar ciertos códigos fuente o un repositorio completo para una arquitectura específica y no posee una herramienta que agilice este proceso realizándose manualmente. Tomando como punto de partida esta problemática, se define como objetivo de la presente investigación implementar el proceso de compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova.

En la actual investigación se recogen los resultados de las herramientas y técnicas empleadas, así como los diferentes elementos de la metodología de desarrollo AUP en su variación UCI y se presenta una herramienta informática que será capaz de lograr la compilación de paquetes de código fuente en la distribución cubana GNU/Linux Nova, de forma distribuida, para mejorar y agilizar este proceso y lograr un producto completamente genuino, contribuyendo así a la necesidad de lograr la independencia tecnológica.

Palabras clave: Compilación, Distribuida, GNU/Linux, Paquetes.

ÍNDICE

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Las computadoras en un inicio ejecutaban instrucciones consistentes en códigos numéricos que señalaban a los circuitos de la máquina los estados correspondientes a cada operación, lo que se denominó lenguaje máquina, luego surgió la necesidad de escribir las instrucciones mediante claves más fáciles de recordar, que al final se traducían al lenguaje máquina. Actualmente los sistemas de GNU/Linux se ocupan de esa complejidad, utilizando paquetes para almacenar todo lo que un programa en particular necesita para su ejecución, evitando los problemas de dependencias perdidas. Un paquete entonces es esencialmente una colección de archivos construidos en uno solo, que ofrece la posibilidad de guardar y proteger el producto durante su distribución y manipulación (Quevedo, 2012).

Los paquetes normalmente contienen todos los archivos necesarios para implementar un conjunto de órdenes o características relacionadas. Hay dos tipos de paquetes: los paquetes binarios y los paquetes fuente. Los paquetes binarios son los que poseen los componentes necesarios listos para instalarse en el Sistema Operativo (SO), y los paquetes fuentes son los que contienen los elementos necesarios para construir un paquete binario (Lic.Pérez, 2013).

Para el trabajo con los paquetes en GNU/Linux es necesario hacer el proceso de compilación de manera constante. La compilación no es más que el proceso de traducción de programas de alto nivel a código máquina. La entrada de este proceso de traducción se denomina programa fuente y el resultado se denomina programa objeto (Quevedo, 2012).

Compilar un programa en GNU/Linux puede tener algunas complicaciones puesto que cada programa tiene sus particularidades, así como que cada computadora o arquitectura necesita de una compilación específica para su tipo. Para un programador o una compañía de software que intentan hacer que un producto llegue a la mayor audiencia posible, esto representa la necesidad de tener múltiples versiones del código fuente para la misma aplicación, lo cual da como resultado una mayor cantidad de tiempo empleada en el mantenimiento del código fuente y problemas adicionales cuando son liberadas las actualizaciones.

Actualmente en Cuba se hace necesario el uso del software libre y por transitividad es necesario realizar un proceso constante de compilación. Bajo la necesidad del uso del software libre se desarrolla la Distribución Cubana de GNU/Linux Nova en el Centro de Software Libre (CESOL) de la Facultad 1 de la Universidad de las Ciencias Informáticas (UCI). La Distribución Cubana de GNU/Linux Nova provee una línea de productos y servicios de calidad orientado a usuarios nacionales y extranjeros que se desempeñan en el área de las tecnologías de software libre, con el objetivo de alcanzar finalmente un SO altamente confiable y que cumpla con las necesidades actuales de Cuba.

Nova es un SO que utiliza el núcleo Linux e incluye determinados paquetes de aplicaciones informáticas para satisfacer las necesidades de la migración a plataformas de código abierto que experimenta Cuba como parte del proceso de informatización de la sociedad. Su proceso de construcción, distribución y mantenimiento está enfocado a alcanzar niveles de excelencia en los siguientes aspectos (Pierra , y otros, 2015):

- Seguridad: el modelo de desarrollo colaborativo, el acceso al código fuente y el exhaustivo proceso de revisión y auditoría de código garantiza un sistema seguro de virus y sin puertas traseras.
- Soberanía Tecnológica: mediante la formación de recursos humanos capacitados será un SO independiente, con capacidad decisional sobre las tecnologías reutilizadas y desarrolladas.
- Socio-adaptabilidad: será un SO hecho por cubanos para cubanos, alineado a las políticas que orienta la informatización nacional y optimizada para las condiciones tecnológicas del País.
- Sostenibilidad: mantendrá un proceso flexible y versátil, en constante innovación y consonancia con las nuevas tendencias tecnológicas internacionales, garantizando modelos de comercialización que permitan el ingreso de divisas por el concepto de exportación de productos y servicios.

En el departamento de SO donde se desarrolla la distribución cubana GNU/Linux Nova se pretende reutilizar las aplicaciones libres que se proveen para otras distribuciones de GNU/Linux, adaptándolas según las necesidades del país. La única forma de garantizar que los binarios que se distribuyen en Nova corresponden con el código fuente que se posee es compilando en un sistema propio.

A medida que maduraba GNU/Linux Nova se hizo evidente que una distribución basada en código fuente no era adecuada para el uso de usuarios finales no especializados en informática. Por lo que se pasó a la conformación de repositorios de paquetes binarios basados en el código fuente aún obtenido desde los repositorios de *Gentoo*[1] y modificado en el seno del proyecto. Dada esta nueva circunstancia, se hizo necesario lograr la construcción de todos los paquetes que se mantenían en el árbol de *Portage*[2]. Fue la primera vez que se pudo comprobar el largo período de tiempo que se requiere para compilar todo un repositorio de paquetes desde código fuente.

[1]*Gentoo*: es una distribución de GNU/Linux
[2]*Portage*: es el gestor de paquetes oficial de la distribución de Linux Gentoo

Al enfrentarse a esta dificultad se habla por primera vez en la tesis de Maestría de Dariem Pérez Herrera uno de los desarrolladores de Nova, del término “compilación distribuida”. La compilación distribuida consiste en una serie de máquinas dedicadas a realizar procesos de compilación comandadas por una serie de nodos denominados maestros, que envían una serie de instrucciones de compilado a la máquina remota para que sean ejecutadas, retornando el resultado de dicho proceso de compilación (Arroyo, 2015). En este punto fue la primera vez que se pensó en usar compilación distribuida para aprovechar al máximo todo el poder de cómputo con que se contaba en el proyecto, al integrar todas las estaciones de trabajo al proceso de compilación de paquetes.

Por otro lado, poder compilar todo el código fuente que integra a una distribución tiene una ventaja muy importante: la posibilidad de optimizar los elementos que la componen, tanto al nivel del código binario de ejecutables y de las bibliotecas, como al nivel de compresión con que se empaqueten. Teniendo en cuenta que los componentes y aplicaciones de una distribución de GNU/Linux por lo general se publican en la red, esto contribuye al uso óptimo del espacio de almacenamiento y del ancho de banda, algo que también influye en la socio-adaptabilidad. A pesar de que desde el año 2015 Pérez Herrera plasmó la necesidad de la implementación de un sistema distribuido para el proceso de compilación, en el departamento de SO aún no se tiene en cuenta su implementación.

Actualmente en el departamento de sistemas operativos de CESOL se cuentan con 30 computadoras: 20 del tipo i3 de 4 GB de RAM y 10 del tipo core 2 DUO de 2 GB de RAM, de estas 30 computadoras solo se usan 8 para el proceso de compilación que son las que utilizan los desarrolladores. Cuando un desarrollador compila un paquete de código fuente lo hace de manera local, o sea directamente de su estación de trabajo. Por lo que si un desarrollador que tenga una computadora I3 de 4 GB de RAM y una conexión de red a 100 MB/s desea compilar un paquete como *Gnome_Shell* demoraría 14 minutos, pero si el procesador es menos fuerte este tiempo puede aumentar desmesuradamente. Por otra parte, existen paquetes que demoran más que otros en ser compilados debido al tamaño de su código fuente, pudiendo demorarse un día entero en el proceso de compilación de un solo paquete, lo que provoca que el desarrollador no puede realizar otra tarea mientras la compilación está en curso. Todo esto también retrasa el proceso de compilación del paquete en conjunto, teniendo que esperar más tiempo hasta que se terminen cada una de las compilaciones. Además, el procesador usa un 100% de su potencia con todos sus núcleos al máximo. Por lo que el consumo y el calor se elevan, haciendo que disminuya la vida útil de la computadora. Sin embargo, al realizar un análisis del equipamiento con el que cuentan en el departamento de SO se puede evidenciar que solo 8 computadoras

intervienen en este proceso, quedando 22 sin que se exploten sus capacidades de procesamiento al máximo.

Dada esta **situación problemática** se define el siguiente **problema científico**: ¿Cómo agilizar el proceso de compilación de paquetes de código fuente en la distribución cubana GNU/Linux Nova?

Se define como **objeto de estudio** el proceso de compilación de paquetes de código fuente en GNU/Linux, teniendo como **campo de acción** el proceso de compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova.

Se trazó como **objetivo general** implementar un sistema de compilación distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova.

El objetivo general se divide en los siguientes **objetivos específicos**:

1. Elaborar los referentes teóricos-metodológicos que sustentan el proceso de compilación distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova.
2. Diseñar una herramienta informática que permita la compilación distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova.
3. Implementar la herramienta informática de compilación distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova.
4. Validar la herramienta informática de compilación distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova mediante la aplicación de pruebas de software.

Se definen como **preguntas científicas:**

1. ¿Cuáles son los presupuestos teóricos que fundamentan el proceso de compilación distribuida de la distribución cubana GNU/Linux Nova?
2. ¿Qué aspectos se deben tener en cuenta para diseñar una herramienta informática para la compilación distribuida de la distribución cubana GNU/Linux Nova?
3. ¿Cuáles son las tecnologías más adecuadas para implementar la herramienta informática para la compilación distribuida de la distribución cubana GNU/Linux Nova?
4. ¿Qué métodos, técnicas y pruebas aplicar para la evaluación de la herramienta informática para la compilación distribuida de la distribución cubana GNU/Linux Nova?

Para el desarrollo de la investigación se emplearon los siguientes **métodos de la investigación científica:**

Los **métodos teóricos** utilizados son:

- **Histórico-Lógico:** para llegar a soluciones similares a la propuesta en este trabajo investigativo.
- **Analítico-sintético:** mediante este método se identifican conceptos y definiciones importantes relacionadas con el tema, permitiendo generar una propuesta adecuada a la situación planteada.
- **Modelación:** se utilizó en el modelado de la propuesta de solución durante la fase de análisis y diseño.

El **método empírico** utilizado es:

- **Entrevista:** para obtener información significativa sobre las necesidades funcionales que debe cumplir la propuesta de solución. Esta técnica se aplicó en el Centro de Software Libre *(Anexo 1).*

El presente trabajo de diploma consta con una introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos.

Capítulo 1: Fundamentación teórica sobre herramienta de compilación distribuida

Explica los principales conceptos que se tratan para lograr un mayor entendimiento del tema tratado. Se hace un estudio de las diferentes herramientas existentes para la compilación de paquetes de código fuente, así como las características de un sistema distribuido para identificar funcionalidades que se le puedan agregar a la solución propuesta.

Capítulo 2: Análisis y diseño de la herramienta de compilación distribuida

Se definen los procesos necesarios del desarrollo de la aplicación, así como el diseño de la estructura del producto, para empezar la implementación según la metodología trazada.

Capítulo 3: Implementación y pruebas de la herramienta de compilación distribuida

Se explican las tareas de implementación, los resultados obtenidos, se exponen las funcionalidades alcanzadas en el período de implementación y se diseñan, realizan y documentan las pruebas para lograr la verificación del producto.

CAPÍTULO 1. Fundamentación teórica sobre herramientas de compilación distribuida

El objetivo de este capítulo consiste en plantear algunos aspectos teóricos que servirán de soporte para la elaboración del sistema. Se exponen conceptos para lograr una mejor comprensión sobre los términos tratados en el proceso de compilación de paquetes, así como un estudio del arte sobre los sistemas de compilación en otras distribuciones de GNU/Linux. Por último, se presentan tanto la metodología como las herramientas y tecnologías a utilizar en la implementación del sistema.

1.1 Conceptos fundamentales

En la presente investigación se exponen los conceptos fundamentales que intervienen en el proceso de compilación de paquetes de código fuente. Este estudio permite una mejor comprensión del objeto de estudio.

1.1.1 Dependencia de paquetes

Las dependencias de un paquete, son aquellos paquetes binarios de los que este depende para poder funcionar en determinada plataforma de hardware y software. Existen dos tipos de dependencias, las de tiempo de construcción *(build dependency)* y las de tiempo de ejecución *(runtime dependency)*. Las dependencias de construcción son aquellos paquetes binarios que se requiere que estén instalados para poder llevar a cabo el proceso de construcción de un paquete binario a partir de un paquete fuente. Las dependencias de tiempo de ejecución son aquellos paquetes que se requieren que estén instalados en el SO para que determinada aplicación, una vez instalada, pueda ejecutarse correctamente (Pérez, 2013).

1.1.2 Repositorio de paquetes

Un repositorio es una estructura organizativa similar a las bases de datos que contiene los paquetes similares entre sí, ya sea binario o fuente, así como la información de los mismos, que están disponibles para ser descargados e instalados (Pérez, 2013). Los repositorios oficiales contienen las aplicaciones que las distribuciones soportan y que dependiendo de sus políticas muchas veces cuentan con un protocolo de revisión rigoroso para asegurarse que todos los paquetes estén en estado óptimo.

1.1.3 Compilación

La compilación es un proceso a través del cual se genera código binario, ejecutable por la CPU[3] de una computadora, a partir de código fuente escrito en algún lenguaje de programación de alto nivel. Por lo general se llama compilación a un proceso que en realidad consta de dos etapas: la primera es la compilación como tal, y es en la que se genera el código objeto específicamente relacionado con un código fuente que puede estar haciendo llamadas a funciones que se encuentran en bibliotecas externas, las cuales se pueden encontrar previamente compiladas, sin embargo, esta primera etapa sólo se concentra en el procesamiento del código fuente en cuestión y no del código externo; mientras que la segunda etapa, que es la de enlace, consiste precisamente en enlazar todas las referencias a funciones externas con la definición de las mismas en las bibliotecas externas requeridas para el funcionamiento del código objeto que se esté generando (Pérez, 2013).

1.1.4 Sistema distribuido

Un sistema distribuido está formado por un conjunto de computadoras unidas por una red de comunicaciones y equipadas con software de sistemas distribuidos. El software de sistemas distribuidos permite a las computadoras coordinar sus actividades y compartir los recursos del sistema: el hardware, el software y los datos. Los usuarios de un sistema distribuido bien diseñado deberían percibir una única facilidad de computación, aún cuando dicha facilidad podría estar formada por un conjunto de computadoras localizadas de manera dispersa (Álvarez, 2015).

1.1.5 Demonio del sistema

Es un programa o proceso que constantemente está disponible y permanece inactivo hasta que es invocado para cumplir una tarea. Un demonio luego puede distribuir la tarea recibida a otros programas o procesos(Pérez, 2013).

1.1.6 Paquete binario

Es un paquete en el cual los componentes que contiene están listos para instalarse en el SO. En el caso de las aplicaciones desarrolladas en lenguajes que requieren ser compilados, el paquete contiene los archivos en algún formato binario compatible con el cargador de GNU/Linux, generalmente en formato ELF[4], además

[3]CPU: unidad central de procesamiento.
[4]ELF (Formato Ejecutable y enlazable): archivos binarios de Linux.

de otros archivos auxiliares como pudieran ser archivos de imágenes o de configuración. En un paquete binario los archivos a instalarse están organizados siguiendo la misma estructura con que serán instalados en el sistema de archivos del SO. En el caso de las distribuciones basadas en la distribución Debian GNU/Linux como la que da origen a este trabajo, se refiere a archivos que tienen como extensión *.deb* o *.udeb* (Guerrero , y otros, 2012).

1.1.7 Paquete fuente

Se conoce como paquete fuente o paquete de código fuente a un paquete que contiene los elementos necesarios para construir a un paquete binario. En el caso de los paquetes fuentes que contienen aplicaciones libres o de código abierto, estos contienen el código fuente de las mismas, así como el resto de los recursos que estas puedan requerir para su compilación, incluyendo imágenes, archivos de configuración, *scripts* para la creación de bases de datos. En el caso de las distribuciones basadas en Debian GNU/Linux, los paquetes fuentes conforman un grupo de archivos: un archivo es el que trae empaquetado el código fuente original desarrollado por el autor original del programa o aplicación; otro contiene las modificaciones o parches del mantenedor o desarrollador de Debian. Este segundo paquete contiene los datos y *scripts* con las reglas requeridas por las herramientas de empaquetado, construcción e instalación de paquetes. Además, se provee un archivo de descripción que contiene información sobre el conjunto de archivos que conforman al paquete fuente, así como firmas de verificación de integridad de los mismos, datos del mantenedor, para qué versión de la distribución de GNU/Linux están empaquetados y demás. Las extensiones que por lo general tienen estos paquetes son las siguientes: el paquete con el código fuente original tiene como extensión .orig.tar.gz o .orig.tar.bz2; el archivo con las modificaciones del desarrollador de la distribución de GNU/Linux se provee con las extensiones .diff.gz o .diff.bz2 para las versiones antiguas del estándar de empaquetado, y .debian.tar.gz o .debian.tar.bz2 para la versión moderna. El archivo de descripción del paquete fuente lleva como extensión .dsc (Lic.Pérez, 2013).

1.2 Herramientas de compilación distribuida de paquetes

Existen varias herramientas que se encargan de hacer la compilación a los paquetes con sus dependencias. El estudio de algunas herramientas de compilación, hizo posible que el equipo de desarrollo del Sistema de Compilación Distribuida de Nova tenga una visión más clara de las funcionalidades que se puedan aprovechar y otras que se pueden mejorar, a la hora de la elaboración de la solución propuesta.

1.2.1 Distcc

Distcc es una herramienta para acelerar la compilación de código fuente a través del uso de computación distribuida sobre una red de cómputo. Con la configuración correcta, *distcc* puede reducir dramáticamente el tiempo de compilación de un proyecto. Está diseñado para trabajar con el lenguaje de programación C (y sus derivados, como C++ y *Objective-C*) y usa a GCC como su respaldo, aunque provee diferentes grados de compatibilidad con Intel C++ *Compiler y Sun Studio Compiler Suite de Sun Microsystems*. Distribuido bajo los términos de la Licencia Pública General de GNU (GNU GPL), distcc es software libre. *Distcc* está diseñado para acelerar la compilación tomando ventaja del poder de procesamiento en desuso en otras computadoras. Una máquina con *distcc* instalado puede enviar código para ser compilado a través de la red hacia otra computadora que tenga el demonio *distccd* y un compilador compatible instalados (Pérez, 2013).

Distcc trabaja como un agente para el compilador. Un demonio *distccd* tiene que correr en cada una de las máquinas participantes. La máquina de origen invoca un preprocesador para manejar los archivos de cabecera, las directivas de preprocesamiento (como es #ifdef) y los archivos fuentes, y envía los códigos fuentes preprocesados hacia otras máquinas sobre la red vía TCP, ya sea sin cifrar o usando SSH. Las máquinas remotas compilan esos archivos fuentes sin ningunas dependencias locales (como son bibliotecas, archivos de encabezado, o definiciones de macros) a archivos objetos y los envían de regreso al origen para continuar con la fase de enlace. La versión 3 de *distcc* soporta un modo (llamado modo surtidor) en el cual los archivos de encabezado incluidos son enviados a las máquinas remotas, permitiendo que el preprocesamiento sea distribuido también. *Distcc* puede trabajar de forma transparente con *ccache*, *Portage*, y *Automake* con una pequeña configuración. Utilizar *distcc* disminuye el tiempo de compilación de una aplicación en específico, pero no disminuye la espera por satisfacción de dependencias (Pérez, 2013).

Distcc siempre debe generar los mismos resultados que una compilación local, es simple de instalar y usar, es más rápido que una compilación local. A diferencia de otros sistemas de compilación distribuida, *distcc* no requiere que todas las máquinas compartan un sistema de archivos, tengan relojes sincronizados, las mismas bibliotecas o archivos de encabezado instalados. Las máquinas pueden ejecutar diferentes sistemas operativos, siempre que tengan formatos binarios compatibles o compiladores cruzados. De forma predeterminada, *distcc* envía el código fuente preprocesado completo a través de la red para cada trabajo, por lo que requiere de las máquinas de voluntarios es que estén ejecutando el demonio *distccd*, y que tienen un compilador apropiado instalado (github, 2017).

La funcionalidad *distcc* "*pump*", agregada en *distcc* 3.0, mejora el *distcc* al distribuir no solo la compilación sino también el preprocesamiento a los servidores de *distcc*. Esto requiere que el servidor y el cliente tengan los mismos encabezados de sistema (el cliente se responsabiliza de la transmisión de encabezados específicos de la aplicación). Dado que, *distcc* en modo de *pump* produce los mismos resultados que *distcc* sin modo *pump*, pero más rápido, ya que el preprocesador no se ejecuta localmente. *Distcc* no es en sí mismo un compilador, sino más bien un *front-end* del compilador GNU C / C ++ (gcc) u otro compilador de su elección. Todas las opciones y características regulares de *gcc* funcionan normalmente (github, 2017).

Distcc está diseñado para ser utilizado con la función de creación paralela de la marca GNU (-j). Enviar archivos a través de la red lleva tiempo, pero pocos ciclos en la máquina del cliente. Todos los archivos que se pueden crear de forma remota son esencialmente "gratis" en términos de CPU del cliente. Esto es aún más cierto en el modo "*pump*", donde el cliente ni siquiera tiene que tomarse el tiempo para preprocesar los archivos fuente. La herramienta *distcc* se ha utilizado con éxito en entornos con cientos de servidores *distcc*, que admiten docenas de compilaciones simultáneas. *Distcc* ahora es razonablemente estable y puede compilar con éxito el *kernel* de Linux, *rsync*, KDE, GNOME (a través de GARNOME), Samba y *Ethereal*. *Distcc* es casi linealmente escalable para pequeñas cantidades de máquinas: para un caso típico, tres máquinas son 2.6 veces más rápidas que una (github, 2017).

1.2.2 Icecc

Icecream (*IceCC*) es un sistema de compilación distribuido para C y C ++, es creado por SUSE y se basa en las ideas y el código de *distcc*. Pero a diferencia de *distcc, icecream* utiliza un servidor central con los horarios de los trabajos de compilación al servidor libre más rápido y dinámico. Utiliza paquetes mantenidos por su distribución, proporciona scripts de inicio personalizados que hace que el *icecream* encaje mejor en la forma en que se configura su sistema. Necesita una computadora que desempeñe un papel de programador ($./*icecc-scheduler* -d), que distribuye el trabajo entre otras computadoras que ejecutan demonios comunes de *IceCC* ($ ./*iceccd* -d). No hay nada que impida que el planificador ejecute un demonio *IceCC* común también. Cada demonio puede comenzar un trabajo de compilación y el planificador se ocupa de una distribución de trabajo óptima entre los demonios disponibles. *IceCC* comprueba la compatibilidad del compilador entre demonios y, de ser necesario, prepara de forma transparente un archivo tar con el entorno de compilación con gcc como compilador predeterminado. Es posible ejecutar el programador y el

demonio en una máquina y solo el demonio en otra, formando así un clúster de compilación con dos nodos (github, 2017).

Plataformas compatibles

La mayoría de los *icecream* son específicos de UNIX y se pueden usar en la mayoría de las plataformas, pero como el programador necesita conocer la carga de una máquina, existen algunas partes complicadas. Soportando las siguientes:

- Linux

- FreeBSD

- DragonFlyBSD

- OS X

Puertos de icecream

TCP / 10245 en las computadoras *demonio* (requerido)

TCP / 8765 para la computadora programadora (requerida)

UDP / 8765 para la difusión para encontrar el planificador (opcional)

1.2.3 Koji

Es el sistema de construcción de paquetes en formato *.rpm* de la distribución Fedora, es una herramienta distribuida donde los desarrolladores utilizando un cliente para solicitar la construcción de un paquete y obtener información sobre la compilación. La principal vista para interactuar con la herramienta es una aplicación web la cual, en lo fundamental, es de solo lectura. Para esto usa una arquitectura de diferentes servidores de centros y constructores llamados Koji demonio. A pesar de su naturaleza poderosa, es bastante difícil de ingresar debido a que requiere muchas competencias diferentes del mantenedor. Para aliviar esto, el proyecto Fedora alberga un servidor público de construcción Koji. Sin embargo, requieren un propio sistema de construcción que sería mantenido por ellos mismos internamente para garantizar la seguridad y controlar la calidad no solo de los resultados sino también de las entradas (Guerrero , y otros, 2012). Hay cinco componentes principales en Koji. Aparte de estos cinco componentes importantes de Koji

también incluyen una base de datos para realizar un seguimiento de las tareas y un almacenamiento compartido para los resultados de compilación.

Componentes de Koji

•**Koji-Hub:** es el centro de todas las operaciones de Koji. Se trata de un servidor XML-RPC, se ejecuta en mod_python en Apache; es pasivo, ya que sólo recibe llamadas XML-RPC y se basa en los demonios de la construcción y otros componentes para iniciar la comunicación; es el único componente que tiene acceso directo a la base de datos y es uno de los dos componentes que tienen acceso de escritura al sistema de archivos.

•**Kojid:** es el demonio de construcción que se ejecuta en cada una de las máquinas de compilación. Su responsabilidad primaria es la manipulación para las solicitudes entrantes de construcción de forma adecuada. Esencialmente kojid pide a koji-hub por trabajo. La creación de imágenes de instalación es también responsabilidad de kojid, utiliza maqueta para la construcción y crea un buildroot nuevo para cada generación, está escrito en *Python* y se comunica con koji-hub a través de XML-RPC.

•**Koji-Web:** es un grupo de scripts que envía correos en *mod_python* y utiliza el motor de plantillas *Cheetah* para proporcionar una interfaz web a Koji, actúa como un cliente de koji-hub, ofrece una interfaz visual para preformas de una cantidad limitada de la administración, expone una gran cantidad de información y también proporciona un medio para determinadas operaciones, como cancelar las generaciones.

•**Koji-client:** es una interfaz de línea de comando escrita en *Python,* permite al usuario consultar la mayor parte de los datos, así como realizar acciones como agregar usuarios y el inicio de construcción de las solicitudes.

• **Kojira:** es un demonio que mantiene la base de construcción actualizada, es responsable de la eliminación de la existencia de una base redundante y de limpiar después de que una solicitud de construcción se ha completado.

Para construir paquetes RPM, Koji usa *Mock. Mock* es una herramienta que construye binarios paquetes de paquetes fuente en un entorno *chroot*. Estos paquetes son entonces almacenados en el caché interno de Koji. Si un usuario quisiera publicar estos paquetes, una herramienta llamada *Mash* se puede utilizar para crear automáticamente un repositorio de paquetes creado con Koji. El usuario necesita usar Koji para etiquetar primero los paquetes deseados y entonces *Mash* encontrará esos paquetes en el caché de Koji y

luego construirá un nuevo RPM repositorio completo con repositorio fuente también. La pronunciada curva de aprendizaje de Koji proviene de los requisitos para el mantenedor.

Con el fin de configurar un servidor de compilación Koji privado, se requeriría un mantenedor para al menos saber cómo crear y mantener certificados SSL, bases de datos postgresql, configuraciones de apache y tener una comprensión básica de cómo yum, simulacro y Koji trabajan juntas, solo por nombrar algunas [KSe]. Los mantenedores también tendrían para configurar todos los diferentes componentes de Koji. Especialmente configurando el Kojid la escalabilidad va a tomar más esfuerzo ya que Koji no ofrece herramientas de escalabilidad para este tipo de actividad. El proyecto Koji proporciona al mantenedor una documentación sobre cómo se configura el sistema.

1.3 Comparación de las herramientas

Para facilitar el análisis de los sistemas de compilación distribuida estudiados se propone comparar las descritas anteriormente. Se emplea la metodología QSOS (*Qualification and Selection of Open Source software*), disponible bajo los términos de la *GNU Free Documentation License* (Licencia de Documentación Libre) para calificar, seleccionar y comparar el software de código libre y abierto de forma objetiva, trazable y argumentada. El proceso consiste en cuatro etapas: definición, evaluación, clasificación y selección. Se establece un método de calificación de software para cuantificar y medir las posibilidades reales de implantación del software ofreciendo posibilidad de comparación al establecer criterios ponderados, en base a los cuales calificar el software y hacer una selección final de la manera más objetiva y beneficiosa (Ramos, y otros, 2011).

Fase de definición: Se establece el marco de referencia para la búsqueda de la información relacionada con las necesidades existentes en el proyecto de software a desarrollar. El estudio realizado en la investigación sobre aplicaciones informáticas para la compilación distribuida de paquetes en Nova está enmarcado en herramientas basadas en GNU/Linux.

Fase de evaluación: Consiste en realizar una caracterización del software analizado.

Calificación: Consiste en la ponderación de los criterios definidos para realizar la comparación de las aplicaciones informáticas analizadas. En la tabla 1 se describe los criterios establecidos.

Tabla 1.Definición de criterios y ponderación

(Fuente: elaboración propia)

Criterios de análisis	**Puntuación**	
	No cubierto 0	Totalmente cubierto 1
Herramienta de compilación distribuida	No es una herramienta de compilación distribuida	Si es una herramienta de compilación distribuida
Soporte del compilador GCC	No utiliza compilador GCC	Si utiliza compilador GCC
Soporte del compilador G++	No utiliza compilador G++	Si utiliza compilador G++
Soporte del compilador CLANG	No utiliza compilador CLANG	Si utiliza compilador CLANG
Distribuida bajo los términos de la Licencia Pública General de GNU (GNU GPL)	No utiliza la licencia GNU GPL	Si utiliza la licencia GNU GPL
Permitir ver que máquinas están compilando	No permite ver que máquinas están compilando	Si permite ver que máquinas están compilando
Utilización de un planificador para distribuir la compilación	No utiliza un planificador	Si utiliza un planificador
Herramienta de construcción de paquetes	No es una herramienta de construcción de paquetes	Si es una herramienta de construcción de paquetes

Fase de selección: Se realiza la comparación de diferentes softwares analizados mediante los criterios definidos en la fase de calificación. Estos criterios permitieron definir un conjunto de aportes que fueron utilizados para definir requisitos funcionales y características que debe tener el proceso de compilación en la distribución cubana GNU/Linux Nova, así como las limitaciones que poseen estas aplicaciones, tomadas en cuenta en el diseño de la propuesta de solución para evitar errores en el proceso de compilación y fortalecer las funcionalidades de la solución. En el caso de los sistemas de construcción de paquetes que se estudiaron, existen varios inconvenientes para su reutilización. Koji se descarta como sistema debido a que sólo sirve para paquetería .rpm, cuando el objetivo de esta investigación es la utilización de un sistema que soporte paquetería .*deb* – se estudió fundamentalmente para conocer las características de estos sistemas y obtener ideas útiles para la implementación otros similares – . Queda centrar el análisis en *distcc*

e *icecc*. Para esta comparación se aplicó la metodología para la clasificación y selección de software libre y código abierto QSOS. A continuación, se muestra en la tabla 2 el resultado de dicha comparación:

Tabla 2. Tabla comparativa de las herramientas de compilación distribuida

(Fuente: elaboración propia)

Criterios de análisis	Herramientas de compilación distribuida	
	Distcc	**Icecc**
Herramienta de compilación distribuida	1	1
Soporte del compilador GCC	1	1
Soporte del compilador G++	1	1
Soporte del compilador CLANG	1	1
Distribuida bajo los términos de la Licencia Pública General de GNU (GNU GPL)	1	1
Permitir ver que máquinas están compilando	0	1
Utilización de un planificador para distribuir la compilación	1	1
Herramienta de construcción de paquetes	0	0
Total	6	7

Ambos sistemas estudiados son libres y de código abierto. El código fuente de los mismos se puede descargar, utilizar, modificar y redistribuir de acuerdo con las cuatro libertades que propugna el movimiento de software libre. Al realizar la comparación mediante QSOS se puede evidenciar que *icecc* posee mayor ponderación que *distcc*, ya que se basa en las ideas y el código de *distcc*. Pero a diferencia de *distcc* *icecream* utiliza un servidor central con los horarios de los trabajos de compilación. Utiliza paquetes mantenidos por su distribución, proporciona scripts de inicio personalizados que hace que el *icecream* encaje mejor en la forma en que se configura su sistema. Necesita una computadora que desempeñe un papel de programador ($./icecc-scheduler -d), que distribuye el trabajo entre otras computadoras que ejecutan demonios comunes de IceCC ($ ./iceccd -d). Es posible ejecutar el programador y el demonio en una máquina y solo el demonio en otra, formando así un clúster de compilación con dos nodos. Sin embargo,

en la práctica, este sistema está configurado para funcionar perfectamente en la plataforma de hardware con que cuenta sus proyectos de origen, pero no están enfocados al uso por terceras partes, por lo que su funcionamiento dentro de la distribución cubana GNU/Linux Nova no es posible. Otro inconveniente que presentan estos sistemas es que requieren intervención humana para la creación del sistema base o imagen inicial donde se realiza la compilación de los paquetes. Por lo que se concluye que la herramienta icecc es la más adecuada según la comparación realizada, pero para poder ser utilizada dentro de la distribución GNU/Linux Nova es necesario realizar una serie de configuraciones y cambios en la misma.

1.4 Metodología de desarrollo de software

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo. La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito, comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado (Maida, 2015).

1.4.1 Metodología de desarrollo Variación de AUP para la UCI

En el desarrollo de la propuesta de solución se utiliza la metodología de desarrollo de software Variación de AUP para la UCI en su escenario 4, una variante realizada por la Universidad de las Ciencias Informáticas a la metodología ágil AUP (*Agile Unified Process, Proceso Ágil Unificado*) y definida por la universidad como guía para la actividad productiva ya permite estandarizar los diferentes productos de trabajo que se generan en sus centros productivos y se adapta al ciclo de vida definido para los proyectos de la UCI (Rodríguez, 2015). Esta metodología propone las siguientes fases:

- Inicio: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo, costo y decidir si se ejecuta o no el proyecto.
- Ejecución: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura, el diseño, se implementa

y se libera el producto. Durante esta fase el software es transferido al ambiente de los usuarios finales o entregado al cliente junto con la documentación. Además, en esta transición se capacita a los usuarios finales sobre la utilización de la aplicación.

- Cierre: En el cierre se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

El desarrollo de la investigación se centrará en la fase de Ejecución y transitará por las siguientes disciplinas: requisitos, análisis, diseño, implementación, pruebas interna y pruebas de aceptación, definidas en la metodología. Porque la fase de inicio se basa en la gestión de proyecto, estimaciones de tiempo, esfuerzo, costo y decidir si se ejecuta o no el proyecto, en la fase de cierre se analizan tantos los resultados del proyecto como su ejecución y las actividades formales de cierre del proyecto que no se realizan en la presente investigación.

1.5 Lenguaje y herramienta de modelado

En el modelado de la propuesta de solución se utiliza el lenguaje UML, al ser el lenguaje de modelado visual más fácil para este tipo de soluciones y la herramienta *Visual Paradigm* en su distribución libre al poder soportar UML, los mismos se describen a continuación:

1.5.1 Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML*, Unified Modeling Language*) es un lenguaje de modelado visual que permite especificar, construir y documentar artefactos de un software. Se puede usar en las diferentes etapas del ciclo de vida de un proyecto. Incluye conceptos semánticos, notación y principios generales de un sistema. Contiene además construcciones organizativas para agrupar los modelos en paquetes, lo que permite dividir grandes sistemas en piezas de trabajo más simples. Este lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo en una empresa, diseño de la estructura de una organización y el diseño del hardware. Un modelo UML está compuesto por tres clases de bloques de construcción (Hernández, 2015):

- Elementos: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- Relaciones: relacionan los elementos entre sí.
- Diagramas: Son colecciones de elementos con sus relaciones.

1.5.2 *Visual Paradigm*

Visual Paradigm versión 8.0 es una herramienta de modelado que soporta el UML y proporciona asistencia al equipo de desarrollo, durante todo el ciclo de vida de la elaboración de un software: análisis, diseño orientados a objetos, construcción, pruebas y despliegue. Se integra con otras aplicaciones, como herramientas ofimáticas, lo cual aumenta la productividad. Genera código de forma automática, reduciendo los tiempos de desarrollo y evitando errores en la codificación del software. Posibilita la obtención de diferentes informes a partir de la información introducida en la herramienta. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Agiliza la construcción de aplicaciones con calidad y a un menor costo. Posibilita la generación de bases de datos, transformación de diagramas de entidad-relación en tablas de base de datos, así como ingeniería inversa de bases de datos (Quintana, y otros, 2011).

1.5.3. Lenguajes y herramientas de implementación

En la implementación de la propuesta de solución se utilizan los siguientes lenguajes y herramientas:

Lenguaje de desarrollo

Bash (*Bourne again shell*) es un programa informático cuya función consiste en interpretar órdenes. Está basado en el *shell* de Unix y es compatible con POSIX. Fue escrito para el proyecto GNU y es el intérprete de comandos por defecto en la mayoría de las distribuciones de Linux. Su nombre es un acrónimo de *Bourne-Again Shell* (otro *shell bourne*) el *Bourne shell (*sh), fue uno de los primeros intérpretes importantes de Unix, la mayoría de los scripts sh pueden ser ejecutados por *Bash* y sin modificación. *Bash* es una de las *shells* más populares de distribuciones de GNU/Linux, la mayoría de los scripts de configuración están programados en este lenguaje (Gómez, 2015).

Las principales características del intérprete *Bash* son:

• Ejecución síncrona de órdenes (una tras otra) o asíncrona (en paralelo).

• Distintos tipos de redirecciones de entradas y salidas para el control y filtrado de la información.

• Control del entorno de los procesos.

• Ejecución de mandatos interactiva y desatendida, aceptando entradas desde teclado o desde ficheros.

• Proporciona una serie de órdenes internas para la manipulación directa del intérprete y su entorno de operación.

• Un lenguaje de programación de alto nivel, que incluye distintos tipos de variables, operadores, matrices, estructuras de control de flujo, entrecomillado, sustitución de valores y funciones.

• Control de trabajos en primer y segundo plano.

• Edición del histórico de mandatos ejecutados.

• Posibilidad de usar una "*shell*" para controlar el entorno del usuario.

Entorno de desarrollo

Geany es un editor de texto con características básicas de un entorno de desarrollo integrado (IDE). Es un entorno de desarrollo integrado multiplataforma, que se apoya en las librerías GTK (Montano, 2016). Una de las principales ventajas es que el código está disponible bajo los términos de la Licencia Pública General de GNU. Presenta algunas de las siguientes características:

- Permite gestionar proyectos grandes de forma sencilla, cosa que otros complican un poco.
- Permite desarrollar códigos en diferentes lenguajes como: C, Java, Pascal, HTML, CSS, PHP y muchos otros.
- Por defecto proporciona la función de autocompletado. Que es algo que otros editores como *Sublime Text* 3 no hace sin su correspondiente *plugins*. Con esta funcionalidad hay que tener cuidado ya que puede llevar a cometer errores de sintaxis que después resultan complicadas de encontrar. Siendo cuidadosos es más una ayuda que un problema.
- Algo que siempre es de agradecer en un editor es que se le pueden instalar *plugins* para añadirle funcionalidades extras que ayuden a desarrollar los códigos de manera más productiva.
- Como en la mayoría de los editores, el código se puede “plegar” por secciones para tener una vista general de todo lo que llevamos escrito.
- Es un entorno ligero y con una curva de aprendizaje sencilla.
- Colorea el código en función del lenguaje que estemos utilizando. Esto facilita las búsquedas de textos.

1.6 Conclusiones parciales

El estudio de los principales conceptos relacionados con la compilación de paquetes en GNU/Linux permitió tener una idea más clara del objeto de estudio y ayudar en una mejor comprensión de los elementos esenciales a tener en cuenta en la posible solución. La selección de la metodología de desarrollo AUP-UCI permitió tener una guía adecuada del proceso de desarrollo de la solución y fueron seleccionadas las herramientas de desarrollo acorde a la propuesta de solución. El estudio del arte permitió concluir que la herramienta *icecc* es la más adecuada para realizar el proceso de compilación distribuida, para que la misma funcione en Nova es necesario hacer algunas variaciones.

CAPITULO 2. Análisis y diseño de la herramienta de compilación distribuida

En este capítulo se abarca el análisis y el diseño que determinaron los elementos teóricos a tener en cuenta para la herramienta de compilación distribuida de Nova, siguiendo la metodología de desarrollo Variación de AUP para la UCI para llegar a tener un proceso de compilación distribuida para Nova con buena calidad teniendo en cuenta las prácticas, los principios que contribuyan a agilizar el proceso y cumpla con las características que requiere este proceso.

2.1 Descripción del contexto del negocio de la propuesta de solución

El autor de la presente investigación realiza un modelo conceptual para comprender el contexto del negocio de la propuesta de solución. En el mismo se muestra en la Figura # 1, que se expone a continuación:

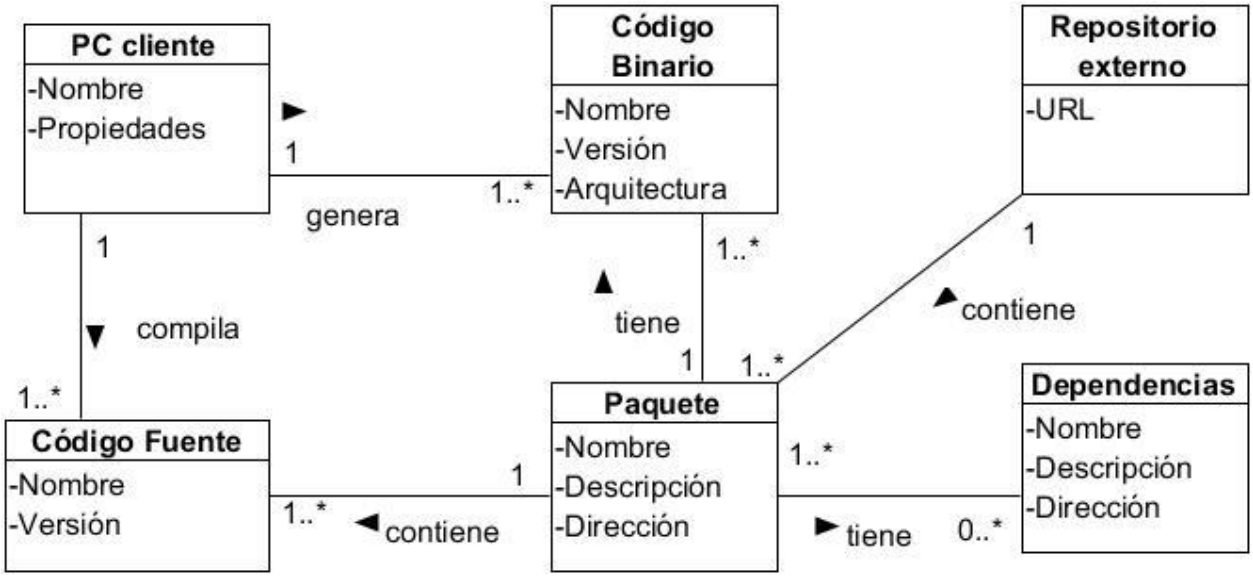


Figura 1. Modelo conceptual

(Fuente: elaboración propia)

Descripción de conceptos:

PC cliente: es la computadora donde se realiza la compilación.

Paquete: Es un archivo comprimido que contiene un conjunto de ficheros con instrucciones para la reconstrucción de una aplicación dentro del sistema nuevo.

Dependencia: Es un paquete accesorio requerido por el paquete de las aplicaciones seleccionadas por el usuario, para hacer funcionar correctamente la compilación.

Repositorio externo: Es la estructura de directorios que contienen los paquetes específicos para la compilación.

Código fuente: es el código al que se realiza la compilación.

Código binario: es el código que se genera después de la compilación.

Actualmente en el departamento de sistemas operativos de CESOL donde se desarrolla la distribución cubana GNU/Linux Nova, se realiza el proceso de compilación de forma local como se muestra en la figura #2. La **PC cliente** descarga el código fuente y las dependencias requeridas del paquete seleccionado de un repositorio externo para realizar el proceso de compilación y generar el código objeto.

2.2 Propuesta de solución

Para darle solución al problema planteado se propone hacer uso de la herramienta *icecc* para realizar el proceso de compilación distribuida dentro de la distribución cubana GNU/Linux Nova como se planteó en las conclusiones del epígrafe 1.3. Con el fin de que funcionara correctamente esta herramienta dentro de Nova fue necesario realizar modificaciones en el archivo *icecc.conf*, este archivo es el que posee todas las configuraciones del sistema. En él es necesario añadir la configuración de los procesos que va a brindar el planificador, quien va a ser el planificador y los datos del mismo, pasándole como parámetro fundamental el ip del planificador. También es necesario adicionar *icecc* al *path* del sistema por lo que es necesario modificar el fichero .*profile*. Con el fin de agilizar el proceso y que *icecc* corra por defecto al encender una computadora se adiciona dicha herramienta al *runs levels* del sistema, lo que permite que inicie automáticamente un *Demonio IceCC*, que es capaz de ubicar automáticamente y conectarse al planificador que se ejecuta en su red utilizando UDP *multi-cast*.

Esta solución hereda de *icecc* la arquitectura cliente servidor que permitirá agilizar el proceso de compilación de paquetes de código fuente en varias computadoras al mismo tiempo con la menor participación posible del usuario. La solución le permitirá a la **PC cliente** descargar el código fuente y las dependencias requeridas para realizar el proceso de compilación al paquete seleccionado de un repositorio externo. Dicha **PC cliente** envía el código fuente preprocesado a una **PC planificador** central con los horarios de los trabajos de compilación al planificador libre más rápido y dinámico encargado de distribuir el código fuente

en pequeños ficheros entre las **PC clientes** con las configuraciones pertinentes que tengan procesadores disponibles para realizar el proceso de compilación. Luego las **PC clientes** envían dichos ficheros compilados a través del planificador a la **PC cliente** que realiza la petición de compilación encargada de enlazar todos los ficheros compilados para generar el código objeto.

En caso de que el usuario desee ver en una interfaz visual la actividad actual en la red de *icecc* lo puede hacer mediante el programa de monitoreo *Icemon* el cual tiene algunas vistas útiles y agradables de ver, como la vista Gantt, que proporciona una visualización satisfactoria de los trabajos actuales que se ejecutan en la red. Lo que facilita la detención de errores en el proceso y poder observar el funcionamiento en conjunto del proceso de compilación distribuida. A continuación, en la Figura # 2 se muestra cómo quedaría de forma gráfica la propuesta de solución:

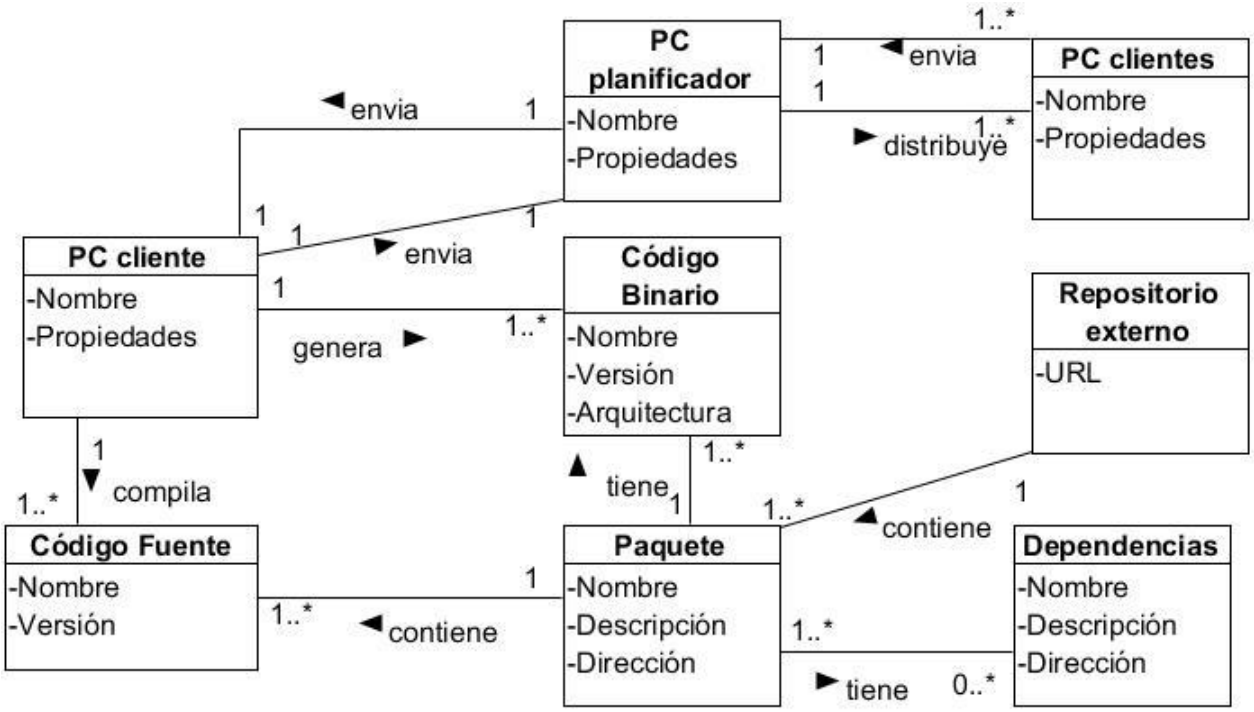


Figura 2.Propuesta de solución

(Fuente: elaboración propia)

2.3 Requisitos de software

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 (Sommerville, 2005).

2.3.1 Técnica de extracción de requisitos

Se usó como técnica de extracción de requisitos la entrevista que es la más tradicional de las técnicas de obtención y consiste en reuniones analista-interesado en las cuales suceden preguntas y respuestas para extraer el dominio de la aplicación (Pressman, 2010).La forma en que se realizó esta técnica fue: el autor de la investigación se entrevistó con el cliente y el jefe del proyecto, a partir de la entrevista realizada detectaron dos requisitos funcionales y once no funcionales (*Anexo 1*).

2.3.2 Requisitos funcionales del sistema

Expresan las capacidades que debe poseer la solución, se identifican a partir de necesidades de negocio o necesidades de los clientes y usuarios. Estos requisitos describen beneficios que recibirán la organización y/o sus clientes con el desarrollo del sistema.

Tabla 3. Requisitos funcionales del sistema

(Fuente: elaboración propia)

Código	Descripción (Requisitos Funcionales)	Prioridad
RF1	Establecer la configuración de la herramienta icecc en el cliente	Alta
RF2	Establecer configuración de la herramienta icecc en el planificador	Alta

2.3.3 Requisitos no funcionales del sistema

Son restricciones de los servicios o funciones ofrecidas por la solución informática (tiempo, proceso o estándares del dominio de negocio). Normalmente afectan al sistema en su totalidad más que una característica o servicio en particular.

Requerimientos de software

RNF1: Se requiere de la distribución de GNU/Linux Nova.

Requerimientos de hardware

RNF2**:** Se requiere de al menos 2 PC.

RNF3: Conexión de red.

RNF4: La memoria RAM de la estación servidor puede tener una capacidad mínima de un 1GB.

RNF5: La memoria RAM de la estación cliente puede tener una capacidad mínima de un 1GB.

RNF6: Debe existir un disco duro para almacenar los paquetes compilados y este puede tener la capacidad mínima de 80 GB.

Requerimiento de Implementación

RNF7: Las normas de codificación se van a llevar a cabo a partir de lo definido por la guía de estilo para el código *Bash*.

RNF8: El sistema debe ser distribuido.

RNF9: Para programar se va a utilizar el lenguaje de alto nivel *Bash*.

RNF10: Para el desarrollo del producto se van a utilizar las herramientas *Geany*.

RNF11: La arquitectura se va a describir bajo el patrón arquitectónico Cliente - Servidor.

2.4 Historias de usuarios (HU)

Las HU son empleadas en la metodología de desarrollo de software Variación de AUP para la UCI para especificar los requisitos. Constituyen una forma de administrar los requisitos sin la elaboración de gran cantidad de documentación y sin requerir mucho tiempo para administrarlos. Estas son escritas mediante la utilización del lenguaje común del usuario por lo que facilita su redacción (Cohn, 205).

Para la descripción de los requisitos funcionales la propuesta de solución que se definió fue una HU para cada uno de ellos para un total de 2 HU. A continuación, se muestran las Historias de Usuario (HU) “Establecer la configuración de la herramienta *icecc* en el cliente” y “Establecer la configuración de la herramienta *icecc* en el planificador”.

Tabla 4. Historia de usuario "Establecer configuración de la herramienta icecc en el cliente"

Historia de usuario	
Número: HU_1	**Nombre:** Establecer la configuración de la herramienta *icecc* en el cliente

Prioridad: Alta	**Iteración Asignada:** 1
Programador: Humberto Pérez Abolaez	**Tiempo Estimado:** 2 meses
Riesgo en Desarrollo: - Fallo de conexión - Fallo del servidor - Interrupción del fluido eléctrico	**Tiempo Real:** 2 meses
Descripción: Será configurada la herramienta icecc que permitirá al cliente utilizar un servidor central con los horarios de los trabajos de compilación al servidor libre más rápido y dinámico. También realizar una petición de compilación al servidor y realizar la compilación de paquetes de código fuente.	

Tabla 5.Historia de usuario "Establecer configuración de la herramienta icecc en el planificador"

Historia de usuario	
Número: HU_2	**Nombre:** Establecer la configuración de la herramienta *icecc* en el planificador
Prioridad: Alta	**Iteración Asignada:** 2
Programador: Humberto Pérez Abolaez	**Tiempo Estimado:** 1 mes
Riesgo en Desarrollo: - Fallo de conexión - Fallo del servidor - Interrupción del fluido eléctrico	**Tiempo Real:** 1 mes
Descripción: Será configurada la herramienta de compilación *icecc* que permitirá al planificador distribuir las tareas de compilación entre los clientes.	

2.5 Descripción de la arquitectura

El estándar IEEE *(Institute of Electrical and Electronics Engineers)* define la arquitectura de software como la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (Garlan, y otros, 1903).Según Roger Pressman:

“En su forma más simple, la arquitectura del software es la estructura u organización de los componentes del programa, la manera en que estos interactúan y la estructura de datos que utilizan” (Pressman, 2009).

Para la presente investigación se propone una arquitectura Cliente-Servidor para la herramienta de compilación distribuida de Nova ya que es la arquitectura que define los componentes del sistema y para seguir con la estructura de los sistemas distribuidos donde un nodo máster, ajustándose a las características del proceso de desarrollo del proyecto, dirige a varios nodos esclavos que se encargan de la compilación de los paquetes.

Esta arquitectura Cliente-Servidor consiste en un modelo en el que dos o más procesos deben colaborar para el progreso mutuo. De estos procesos, para simplificar la explicación, se asume la existencia de únicamente dos, uno suele realizar las tareas más pesadas, al cual se denomina servidor, y otro posee la interacción con el usuario, que será el cliente. Este modelo puede variar de modo que el cliente puede poseer cierta capacidad de cómputo. Se pueden diferenciar tres componentes esenciales (Villar, 2010):

El cliente: el proceso cliente es el que generalmente realiza la interacción con el usuario, la mayoría de veces en forma gráfica. Posee procesos auxiliares que se encargan del establecimiento de conexión y de las comunicaciones con el servidor, así como tareas de sincronización.

El servidor: el proceso servidor, como su nombre indica, proporciona un servicio al cliente, del cual devuelve los resultados. Puede poseer, aunque no siempre, procesos auxiliares que se encargan de las tareas de comunicación con el cliente. Dado que es el que proporciona los resultados al cliente, la carga computacional asociada a éstos es mayor que la de los clientes.

La comunicación: para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, la cual proporciona los mecanismos básicos de direccionamiento y transporte. Las comunicaciones se pueden realizar por medios tanto orientados a la conexión (TCP) como no orientados a la conexión (UDP). La red debe tener características adecuadas de rendimiento, confiabilidad, transparencia y administración.

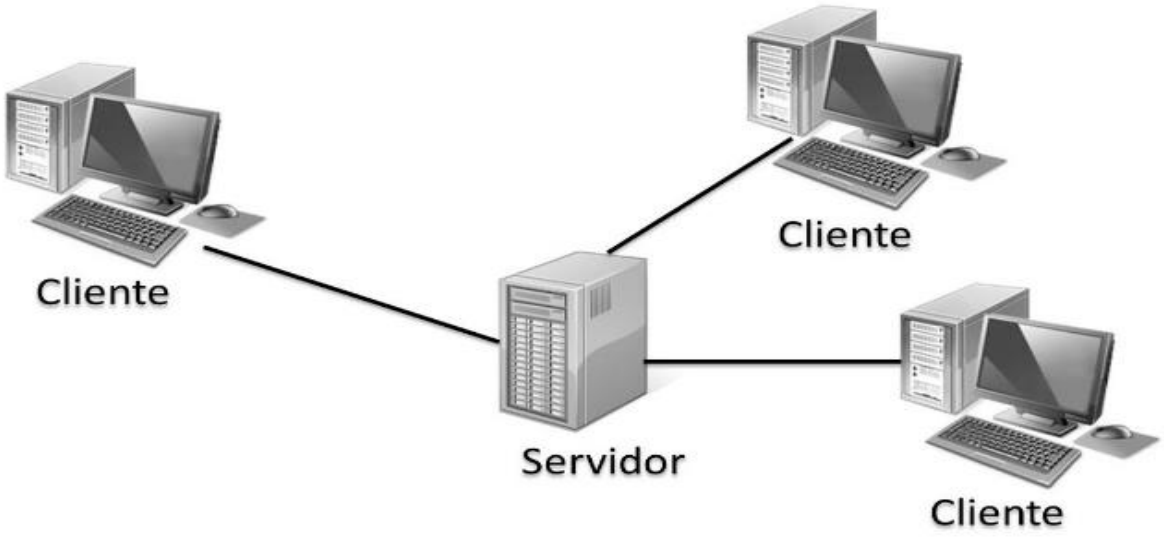


Figura 3.Arquitectura cliente servidor

(Fuente:(Ávila, 2013))

2.6 Diagrama de despliegue

En el diagrama de despliegue indica la situación física de los componentes lógicos desarrollados. Es decir, se sitúa el software en el hardware que debe contener. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos. Un nodo es un recurso de ejecución representado por un cubo, tal como una computadora, un dispositivo, o memoria. Esta vista permite determinar las consecuencias de la distribución y de la asignación de recursos.

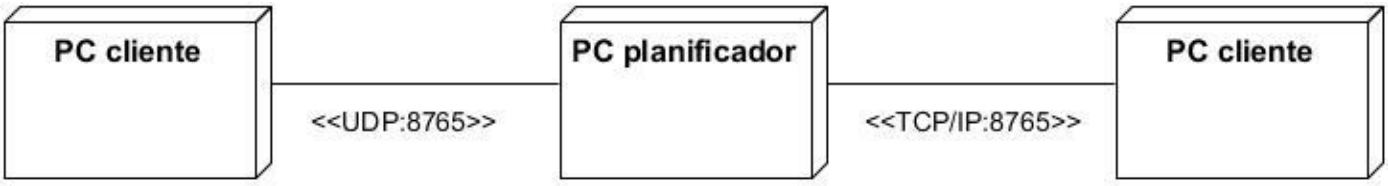


Figura 4.Diagrama de despliegue

(Fuente: elaboración propia)

2.7 Conclusiones parciales

Se lograron extraer mediante la técnica de entrevista 2 requisitos funcionales y 11 no funcionales lo que permitió tener claridad en las características de la propuesta de solución. También se realizaron las descripciones de las historias de usuario, el modelo conceptual, y el diagrama de despliegue como lo

propone la metodología propuesta, lo que permitió tener una guía de la situación real de la propuesta de solución. Se hizo uso de la arquitectura cliente-servidor en la solución que permitió una propicia organización de la herramienta a implementar.

CAPÍTULO 3. Implementación y pruebas de la herramienta de compilación distribuida

Este capítulo centra la atención en la implementación del proceso de construcción de la herramienta de compilación distribuida para la distribución cubana GNU/Linux Nova, además de las pruebas realizadas a la misma. Se exponen los estándares de codificación empleados para el desarrollo de la herramienta y se establece la estrategia de pruebas para verificar la calidad de la herramienta implementada, así como la documentación referente a las pruebas seleccionadas.

3.1 Estándar de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código el cual refleja un estilo armonioso, como si un único programador hubiese escrito todo el código de una sola vez. Cuando el proyecto de software incorpora código fuente previo, o cuando realiza el mantenimiento de un sistema de software el estándar de codificación debería establecer cómo operar con la base de código existente. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento (GONZALO, 2016).

A continuación, se mencionan las principales prácticas utilizadas para la implementación de herramienta de compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova:

Identación: Emplear 4 espacios por cada nivel de identación.

```
configure_scheduler()
{
   echo "Configurado planificador"
   update-rc.d -f icecc-scheduler remove
   update-rc.d icecc-scheduler defaults
   service icecc-scheduler start

}
```

Figura 5.Identación

(Fuente: elaboración propia)

Tabuladores y espacios: No mezclar tabulaciones con espacios.

```
function val_ip()
  {

   ip=$1
   stat=1
  if [[ $ip =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then
        OIFS=$IFS
        IFS='.'
        ip=($ip)
        IFS=$OIFS
        [[ ${ip[0]} -le 255 && ${ip[1]} -le 255  && ${ip[2]} -le 255 && ${ip[3]} -le 255 ]]
        stat=$?
  fi
  return $stat

  }
```

Figura 6.Tabuladores y espacios

(Fuente: elaboración propia)

Tamaño máximo de líneas: Limita todas las líneas a un máximo de 80 caracteres. Esto puede ser realizado mediante el uso de paréntesis de forma implícita o mediante el uso de la barra invertida.

```
configure_client()
{

    if val_ip $1; then
        echo "Configurado cliente"
        np=`nproc`
        sed  's/ICECC_MAX_JOBS=".*"/ICECC_MAX_JOBS="'$np'"/g' /etc/icecc/icecc.conf |
        sed  's/ICECC_SCHEDULER_HOST=".*"/ICECC_SCHEDULER_HOST="'$1'"/g' > /etc/icecc/tmp.conf
        mv /etc/icecc/tmp.conf /etc/icecc/icecc.conf
        exist=`grep -nre 'PATH="/usr/lib/icecc/bin:$PATH"' /etc/profile`
        if [ "x$exist" = "x" ]; then
            sed -e '$a \ \nif [ -d "/usr/lib/icecc/bin" ];
            then \n   PATH="/usr/lib/icecc/bin:$PATH"\nfi' /etc/profile > /etc/tmpprofile
            mv /etc/tmpprofile /etc/profile
            . /etc/profile
        fi
        service iceccd restart
        else
        echo "La direción ip no es correcta";
        print_help;
    fi


}
```

Figura 7.Tamaño máximo de líneas

(Fuente: elaboración propia)

Declaraciones: Se deben declarar cada variable en una línea diferente.

Asignación de nombres: Evitar nombres largos y que difieran en una letra o en el uso de mayúsculas. Para nombrar funciones y variables se describe con la primera palabra en minúscula.

3.2 Pruebas de software Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan la excelencia, el desempeño de un software, involucra las operaciones del sistema bajo condiciones controladas y evalúa los resultados. Las técnicas para encontrar problemas en un programa son variadas y van desde el uso del ingenio por parte del personal de prueba hasta herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad (Pressman, 2002).

3.2.1 Pruebas a realizar

Funcionales

Se escoge este nivel de prueba ya que lo que se desea es la verificación de las funcionalidades del sistema y su relación con los demás componentes del sistema.

Las pruebas de sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica (Pressman, 2002).

Unitarias

La prueba de unidad se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño del software: el componente o módulo de software. Tomando como guía la descripción del diseño a nivel de componentes, se prueban importantes caminos de control para descubrir errores dentro de los límites del módulo. Las pruebas de unidad se concentran en la lógica del procesamiento interno y en las estructuras de datos dentro de los límites de un componente (Pressman, 2009).

3.3 Método de prueba

Caja Blanca

La prueba de caja blanca, en ocasiones llamada prueba de caja de cristal, es un método de diseño que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba. Al emplear los métodos de prueba de caja blanca, el ingeniero del software podrá derivar casos de prueba que (Pressman, 2009):

1. Garanticen que todas las rutas independientes dentro del módulo se han ejercitado por lo menos una vez.
2. Ejerciten los lados verdadero y falso de todas las decisiones lógicas.
3. Ejecuten todos los bucles en sus límites y dentro de sus límites operacionales.

4. Ejerciten estructuras de datos internos para asegurar su validez.

Caja Negra

Se escoge el método de caja negra ya que los niveles de pruebas escogidos pretenden probarlas funcionalidades del sistema, para esto es necesario este método ya que permite comprobar el comportamiento del software a través de los requisitos funcionales (Pressman, 2002), obviando el funcionamiento interno y la estructura del programa.

Las pruebas de caja negra pretenden encontrar estos tipos de errores:

- Funciones incorrectas o ausentes.
- Errores en la interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y de terminación.

3.4 Técnica de prueba

Partición equivalenteCaracterísticas

- Divide el dominio de entrada de un programa en clases de datos, a partir de las cuales pueden derivarse casos de prueba.
- Descubre clases de errores, que, de otra manera, requeriría la ejecución de muchos casos antes de que se observe el error general.
- Reducir al máximo el total de casos de prueba que deben desarrollarse (Cabeza Chávez, 2015).

El diseño consiste

- Identificar clases de equivalencia.
- Crear los casos de prueba.

Clase de equivalencia

- Conjunto de estados válidos o inválidos para condiciones de entrada. Las clases de equivalencia se definen de acuerdo a las siguientes directrices:

1 Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.

2 Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.

3 Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y otra no válida

4 Si una condición de entrada es booleana, se define una clase de equivalencia válida y otra no válida (Cabeza Chávez, 2015).

Camino básico

La prueba de camino básico es una técnica de prueba de caja blanca. Este método permite que el diseñador de casos de pruebas obtenga una medida de complejidad lógica de un diseño procedimental y que use esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para ejercitar el conjunto básico deben garantizar que se ejecutan cada instrucción del programa por lo menos una vez durante la prueba (Pressman, 2009).

3.5 Aplicación de las pruebas

Pruebas unitarias Las pruebas unitarias fueron realizadas mediante el método de caja blanca mediante la técnica camino básico. A continuación, se realiza la técnica del camino básico, al método *configure_client().*

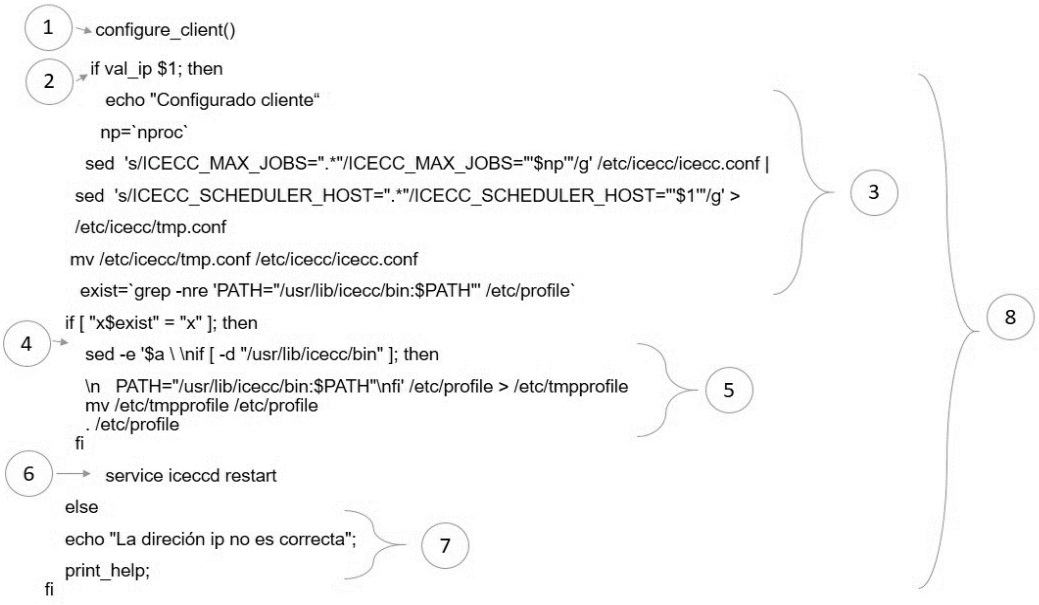


Figura 8.Técnica de camino básico

(Fuente: elaboración propia)

Luego de numerar las líneas de código, se diseña la gráfica del programa que describe el flujo de control lógico empleando nodos y aristas.

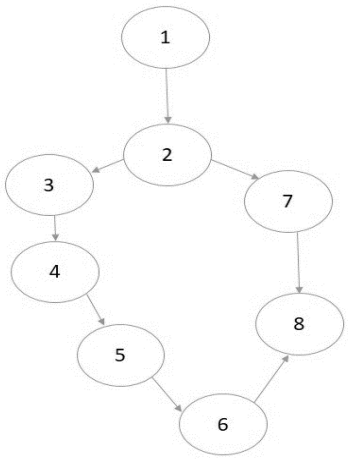


Figura 9.Grafo de flujo

(Fuente: elaboración propia)

A partir del grafo obtenido con 8 nodos y 8 aristas, se calcula la complejidad ciclomática *V (G)* la cual constituye una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa.

V(G) = cantidad_aristas – cantidad_nodos + 2

V(G) = 8 – 8 + 2 = 2

Un camino independiente es cualquier camino del programa que introduce al menos un nuevo conjunto de sentencias de proceso o una nueva condición (Pressman, 2009). La cantidad de caminos independientes se establece por la complejidad ciclomática, por tanto, se identifican 2 caminos como se muestra en la siguiente tabla.

Tabla 6. Listado de caminos independientes

(Fuente: elaboración propia)

No.	Camino
1	1-2-3-4-5-6-8
2	1-2-7-8

El valor de la complejidad ciclomática ofrece además un límite superior para la cantidad de pruebas que se deben diseñar y ejecutar para garantizar que se cumplen todas las sentencias

del programa (Pressman, 2009). A continuación, se muestran los casos de pruebas para cada camino independiente.

Tabla7. Caso de prueba camino 1

(Fuente: elaboración propia)

Caso de prueba de unidad	
No. Camino: 1	**Camino:** 1-2-3-4-5-6-8
Nombre de la persona que realiza la prueba: Humberto Pérez Arbolaez.	
Descripción de la prueba: Verificar configuración del cliente.	
Entrada: Dirección ip: 10.53.4.84	
Resultado esperado: Se muestra el mensaje “Cliente configurado”.	
Evaluación de la prueba: Satisfactoria. Se configuró el cliente correctamente.	

Tabla7. Caso de prueba camino 2

(Fuente: elaboración propia)

Caso de prueba de unidad	
No. Camino: 2	**Camino:** 1-2-7-8
Nombre de la persona que realiza la prueba: Humberto Pérez Arbolaez.	
Descripción de la prueba: Verificar configuración del cliente.	
Entrada: Dirección ip: asd	
Resultado esperado: Se muestra el mensaje de error “La dirección ip no es correcta”.	
Evaluación de la prueba: Satisfactoria. Se mostró el mensaje error correctamente.	

Resultado de las pruebas unitarias

Fueron realizadas las pruebas unitarias a la configuración del cliente mediante el método de caja blanca aplicando la técnica de camino básico donde no se detectaron errores en los caminos que se obtuvo en la funcionalidad. Demostrando una configuración satisfactoria para el cliente.

Pruebas Funcionales

Las pruebas funcionales fueron realizadas mediante el método de caja negra empleando la técnica partición equivalente. A continuación, se muestra el diseño de Caso de Prueba basados en Historias de Usuarios para el caso de prueba “Establecer la configuración de la herramienta *icecc* en el cliente” y “Establecer la configuración de la herramienta *icecc* en el planificador”.

Tabla 8. Condición de ejecución

(Fuente: elaboración propia)

No	Condiciones de ejecución
1	Debe estar instalada la herramienta *icecc*
2	Debe existir al menos un planificador

Casos de pruebas 1. Establecer la configuración de la herramienta *icecc* en el cliente.

Descripción de las variables

Tabla 9. Descripción de las variables del caso de prueba "Establecer la configuración de la herramienta icecc en el cliente."

No	**Nombre de Campo**	**Clasificación**	**Valor Nulo**	**Descripción**
1	Nombre del	*string*	no	Acepta cualquier

	servicio.			carácter alfa-numérico, no debe estar vacío.
2	Dirección ip del servicio	*string*	no	Debe ser una dirección ip válida, no bebe estar vacío.

Tabla 10. Descripción de los escenarios del caso de prueba " Establecer la configuración de la herramienta icecc en el cliente"

Escenario	Descripción	Nombre del servicio	Dirección ip del servicio	Respuesta del sistema	Flujo central
EC 1.1 Configuración del cliente con valores válidos	El sistema permite configurar el servicio correctamente.	V *client*	V 10.53.4.84	Se configura el cliente y el sistema muestra un mensaje de confirmación: "Configurado cliente."	1. Buscar en todas las aplicaciones la terminal y presionar Enter. 2. El usuario se autentica como administrador insertando en la consola el comando sudo su y la contraseña de administrador y presionar Enter. 3. El usuario ejecuta el comando: **bash nova-compiler.sh --configure client --scheduler ip** yluego presionar Enter.
EC 1.2 Configuración del cliente con valores inválidos.	El sistema no permite configurar el servicio.	I *client*	V 10.53.4.84	No configura el cliente y el sistema muestra un mensaje de error: "nova-compiler --configure <client|scheduler>","Example: nova-compiler --configure client --scheduler 192.168.134.34".	
		V *client*	I Fdsfd	El sistema muestra un mensaje de error: "La dirección ip no es correcta."	
EC 1.3 Configuración del cliente con campos vacíos.	El sistema no permite configurar el servicio.	I (vacío)	V 10.53.4.84	El sistema muestra un mensaje de error: "nova-compiler --configure <client|scheduler>","Example: nova-compiler --configure client --scheduler 192.168.134.34".	
		V *client*	I (vacío)	El sistema muestra un mensaje de error: "nova-compiler --configure <client|scheduler>","Example: nova-compiler --configure client --scheduler	

				192.168.134.34".	

Casos de pruebas 2. Establecer la configuración de la herramienta *icecc* en el planificador.

Tabla 7. Condición de ejecución
(Fuente: elaboración propia)

No	Condiciones de ejecución
1	Debe estar instalada la herramienta *icecc*

Descripción de las variables

Tabla 8. Descripción de las variables del caso de prueba "Establecer la configuración de la herramienta icecc en el planificador"

No	Nombre de Campo	Clasificación	Valor Nulo	Descripción
1	Nombre del servicio.	*string*	no	Acepta cualquier carácter alfa-numérico no debe estar vacío.

Tabla 9. Descripción de los escenarios del caso de prueba " Establecer la configuración de la herramienta icecc en el planificador"

Escenario	Descripción	Nombre del servicio	Respuesta del sistema	Flujo central
EC 1.1 Configuración del panificador con valores válidos	El sistema permite configurar el servicio correctamente.	V scheduler	Se configura el planificador y el sistema muestra un mensaje de confirmación: “Configurado planificador.”	1. Buscar en todas las aplicaciones la terminal y presionar Enter. 2. El usuario se autentica como administrador insertando en la consola el comando sudo su y la
EC 1.2 Configuración del cliente con valores	El sistema no permite configurar el servicio.	I asd	No configura el planificador y el sistema muestra un mensaje de error: “nova-compiler --configure <client|scheduler>”,"Example: nova-compiler --configure	

inválidos.			client --scheduler 192.168.134.34".	contraseña de administrador y presionar Enter. 3. El usuario ejecuta el comando: **bash nova-compiler.sh --configure scheduler** y luego presionar Enter.
EC 1.3 Configuración del cliente con campos vacíos.	El sistema no permite configurar el servicio.	I (vacío)	No configura el planificador y el sistema muestra un mensaje de error: “nova-compiler --configure <client|scheduler>”,"Example: nova-compiler --configure client --scheduler 192.168.134.34".	

Resultado de las pruebas funcionales

Para lograr que la salida final de la herramienta de compilación distribuida de la distribución cubana GNU/Linux Nova coincidiera con el resultado esperado por el cliente y para probar cada una de las funcionalidades establecidas en la herramienta se realizaron las pruebas funcionales, divididas en dos iteraciones. En la primera iteración se detectaron 5 no conformidades, de las cuales cuatro son de validación y una de ortografía. En una segunda iteración se realizaron las pruebas funcionales, para los restantes requisitos que se implementaron y para las funcionalidades que resultaron fallidas donde no se detectaron no conformidades, lo que significa que la herramienta funciona correctamente cumpliendo con las expectativas del cliente. A continuación, se muestra como quedan reflejados los resultados por cada una de las iteraciones de pruebas funcionales de caja negra realizadas al sistema:

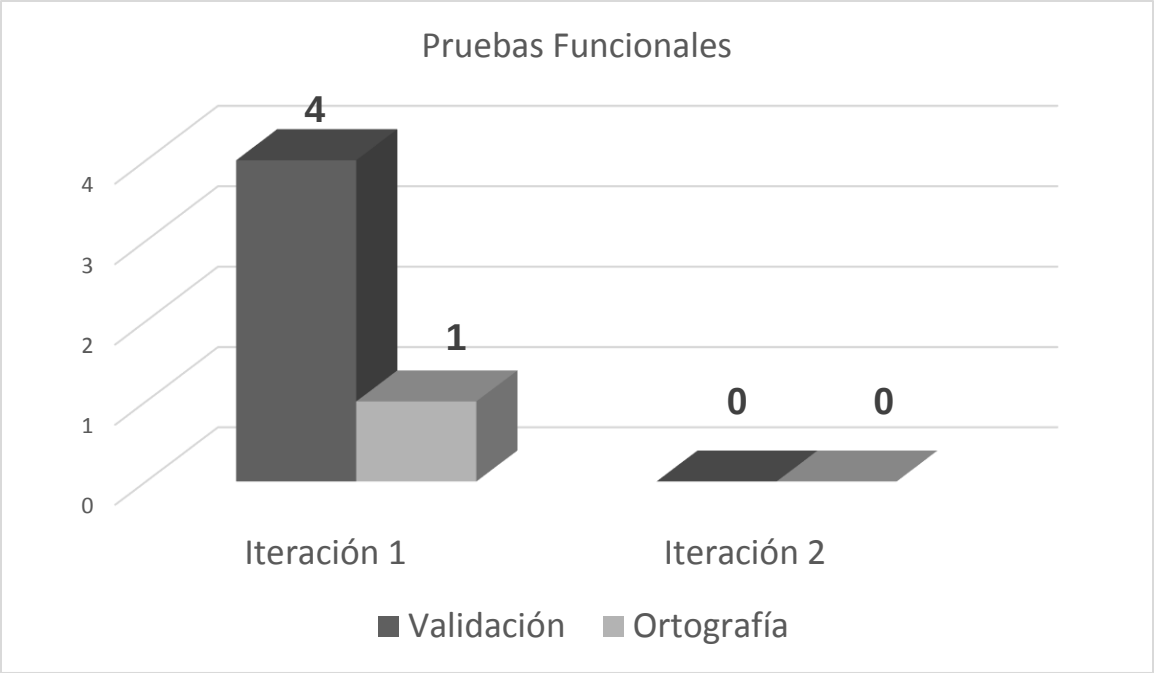


Figura 10.Resultados de las pruebas funcionales

(Fuente: elaboración propia)

3.6 Pruebas de Aceptación

Se escoge realizar la prueba de aceptación ya que es necesario para la validación de la solución que el cliente este de acuerdo con el funcionamiento del sistema desarrollado y así pueda emitir la carta de aceptación donde se demuestra la conformidad del cliente la solución. Como técnica se escoge las pruebas alfas, esta se lleva a cabo por un cliente en el lugar de desarrollo. Estas pruebas se desarrollan en un entorno controlado y se ejecuta el software de forma natural con el desarrollador como observador del usuario.

Para que tengan validez se debe crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto se procede a realizar las pruebas y a documentar los resultados. El resultado de las pruebas de aceptación se muestra en el anexo 3 donde fue emitida el acta de aceptación con el cliente.

3.7 Comparación entre la compilación local y la distribuida

En la actual investigación se realizó una comparación de los tiempos de compilación local y la distribuida en la distribución cubana GNU/Linux Nova con una muestra de paquetes pequeños, medianos y grandes en computadoras con las propiedades siguientes i3 de 4 GB de RAM y una conexión de red a 100 MB/s. En la tabla trece se muestran un aproximado del promedio de tiempo de compilación local y distribuida en los paquetes seleccionado donde se demuestra que la compilación distribuida es mucho más rápida que la local.

Tabla 10. Tiempos de compilación

(Fuente: elaboración propia)

Paquetes	Tiempos de Compilación		Promedio	
	Local	Distribuida	15 seg	4 seg
gnome-calendar	20 seg	6 seg		
a11y-profile-manager	14 seg	4 seg		
hello	11 seg	2 seg		
grub	12 min	4 min	12 min	4 min
gnome-shell	14 min	5 min		
gnome-software	10 min	3 min		
libatk1.0-dbg	20 min	6 min	25 min	8 min
lib32asan2-dbg	25 min	8 min		
language-pack-zh-hant-base	30 min	10 min		

A continuación se muestra un gráfico con los tiempos promedios de la compilación de paquetes.

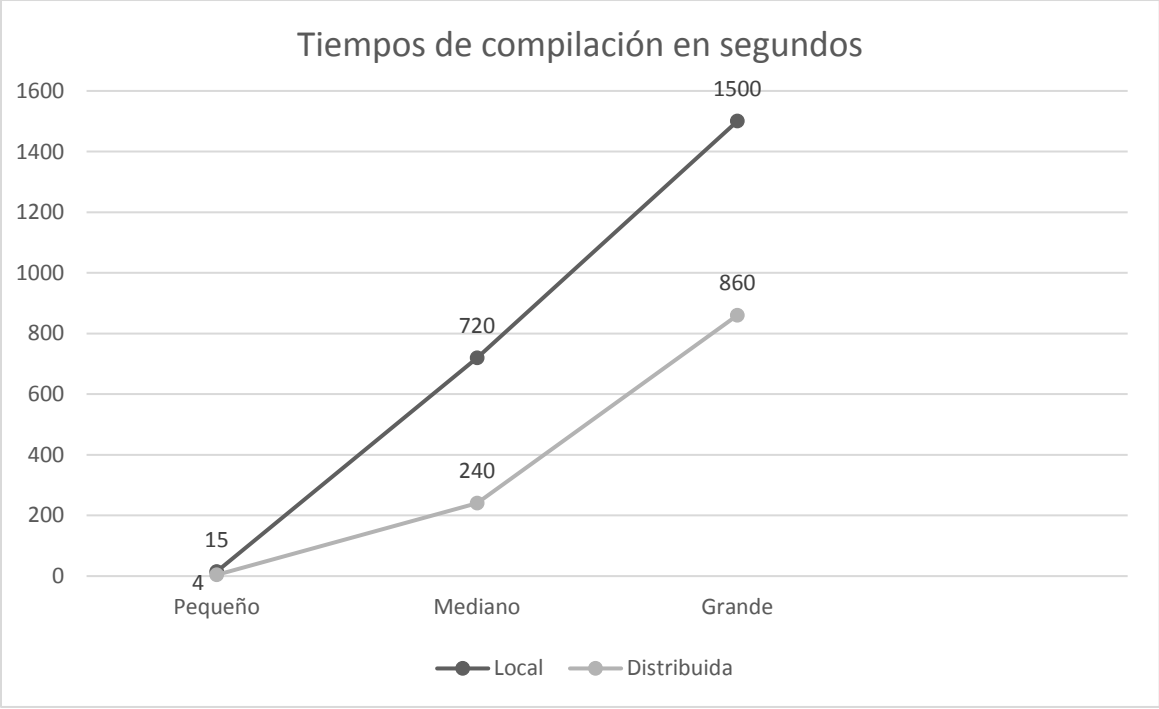


Figura 11.Gráfico de tiempos

(Fuente: elaboración propia)

Como se puede apreciar en la gráfica los tiempos de la compilación distribuida son menores que los tiempos de la local en todos los paquetes, demostrando que entre más grande sea el paquete seleccionado más disminuye el tiempo de compilación distribuida con respecto a la local.

3.8 Evaluación del objetivo general de la investigación

La técnica de *IADOV* se compone de cinco preguntas claves: tres cerradas y dos abiertas, es utilizada para determinar el nivel de satisfacción individual y grupal de los usuarios a partir de una encuesta elaborada según las exigencias pertinentes. En la presente investigación fue aplicada a siete especialistas.

La técnica de *IADOV* constituye una vía indirecta para el estudio de satisfacción, ya que los criterios que se utilizan se fundamentan en las relaciones que se establecen entre las tres preguntas cerradas, que se intercalan dentro de un cuestionario y cuya relación el encuestado desconoce. En este caso, de la encuesta, son las preguntas 2,3 y 4 se relacionan a través de lo que se denomina el “Cuadro Lógico de IADOV” que se muestra en la tabla 14 (

*Anexo **2***).

Tabla 11. Cuadro lógico de IADOV

4. Luego de haber visto la herramienta para agilizar la compilación de paquetes de código fuente en la distribución cubana GNU/Linux Nova refleje en qué medida le gusta la solución desarrollada.	2. ¿Considera usted la forma en que se compilan los paquetes de código fuente durante el desarrollo de la distribución cubana GNU/Linux Nova es ágil?								
	No			**No sé**			**Si**		
	3. ¿Considera usted factible disponer para agilizar la compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova?								
	Si	**No sé**	**No**	**Si**	**No sé**	**No**	**Si**	**No sé**	**No**
Me gusta mucho	1	2	6	2	2	6	6	6	6
Me gusta más de lo que me disgusta	2	2	3	2	3	3	6	3	6
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	4	4
No me gusta nada	6	6	6	6	4	4	6	4	6
No sé decir	2	3	6	3	3	3	6	3	4

La forma de utilizar la tabla es la siguiente: Cada encuestado recibe una evaluación individual en dependencia de las respuestas que dé a las preguntas cerradas. Para facilitar el procesamiento posterior, en el diseño de la encuesta se debe tener en cuenta que a estas preguntas sólo se responda de la forma prevista en el cuadro lógico de IADOV. Las respuestas a las preguntas 2 y 3 pueden ser SI, NO, NO SÉ, y a las preguntas 4, “Me gusta mucho”, “Me gusta más de lo que me disgusta”, “Me da lo mismo”, “Me disgusta más de lo que me gusta”, “No me gusta nada”, o “No sé qué decir”.

Para obtener el índice de satisfacción grupal (ISG) se trabaja con los diferentes niveles de satisfacción que se expresan en la escala numérica que oscila entre +1 y – 1. El número resultante de la interrelación de las tres preguntas que indica la posición de cada encuestado en la siguiente escala de satisfacción:

1. Clara satisfacción +1

2. Más satisfecho que insatisfecho 0.5

3. No definido y contradictorio 0

4. Más insatisfecho que satisfecho -0.5

5. Clara insatisfacción -1

El índice de satisfacción grupal (ISG) se expresa en una escala numérica que va desde 1 (máxima satisfacción), hasta -1 (máxima insatisfacción). El ISG se calcula mediante la siguiente fórmula:

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

En esta fórmula A, B, C, D, E, representan la cantidad de encuestados colocados respectivamente en las posiciones de satisfacción 1; 2; 3 u 6; 4; 5 y donde N representa la cantidad total de encuestados

Por ejemplo, si 4 especialistas presentan máxima satisfacción, 3 están más satisfechos que insatisfechos, el índice de satisfacción se calcularía de la siguiente forma:

$$ISG = \frac{4(+1) + 3(+0.5) + 0(0) + 0(-0.5) + 0(-1)}{7}$$

ISG=0.7

Los valores de ISG que se encuentran comprendidos entre -1 y -0,5 indican insatisfacción; los comprendidos entre -0,49 y +0,49 evidencian contradicción y los que caen entre 0,5 y 1 indican que existe satisfacción.

El proceso de evaluación del objetivo de la investigación mediante la técnica de IADOV confirmó su factibilidad de uso, expresando cuantitativamente en el alto ISG (0.7) y cualitativamente en los criterios emitidos en el centro de investigación CESOL, lo que refleja la aceptación de la propuesta y el reconocimiento a su utilidad.

3.7 Conclusiones parciales

Las realizaciones de las fases definidas en el presente capítulo permitieron obtener como resultado la imagen de configuración de la herramienta de compilación distribuida para la distribución cubana GNU/Linux Nova, siendo la misma el producto que se propone como solución de la presente investigación. Los diseños de los casos de pruebas creados a partir de los requisitos definidos del sistema permitieron validar la propuesta de solución luego de dos iteraciones de pruebas que dieron como resultado un sistema libre de no conformidades. Las pruebas de aceptación realizadas validaron la propuesta de solución por parte del cliente y su satisfacción con el producto final.

CONCLUSIONES

- ✓ El análisis de los referentes teóricos y de las herramientas de compilación distribuidas estudiadas evidenció la necesidad de desarrollar una herramienta de compilación distribuida para agilizar el proceso de compilación en la distribución cubana GNU/Linux Nova.
- ✓ Se diseñó una herramienta informática que permite la compilación distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova en correspondencia con los requisitos definidos con el cliente.
- ✓ Se implementó una herramienta informática que permite la compilación de paquetes de código fuente en la distribución cubana GNU/Linux Nova cumpliendo con los requisitos definidos por el cliente.
- ✓ La evaluación de la investigación se realizó a partir de la aplicación pruebas de software que garantizan el correcto funcionamiento y la técnica de *IADOV* que demostró la satisfacción del cliente hacia la herramienta desarrollada.

RECOMENDACIONES

Configurar una red para múltiples PC, para realizar la configuración de los clientes desde un pc planificador a través de la subred y realizar el proceso de compilación distribuida con los clientes definidos por el planificador.

REFERENCIAS BIBLIOGRÁFICAS

G. Figueroa , Roberth, J. Solís , Camilo y A. Cabrera , Armando. 2007.*METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* 2007.

Pierra , Allan Fuentes y Rodríguez , Héctor Figueredo. 2015. Nova, distribución cubana de GNU/Linux. Reestructuración estratégica de su proceso de desarrollo.Tesis de Maestría. [En línea] 1 de Febrero de 2015. [Citado el: 11 de Octubre de 2017.] https://repositorio.uci.cu/jspui/handle/ident/8710.

Quintana, Yoandri Rondón, Camejo , Lianet Domínguez y Díaz , Abel Berenguer. 2011. Diseño de la base de datos para sistemas de digitalización y gestión de medidas. [En línea] 2011. [Citado el: 6 de 12 de 2017.] http://laboratorios.fi.uba.ar/lie/Revista/Articulos/080815/A3mar2011.pdf.

Álvarez, Fernando García. 2015. Sistema de Distribución de Objetos para un Sistema Distribuido Orientado a Objetos soportado por una Máquina Abstracta. [En línea] 2015. [Citado el: 18 de Noviebre de 2017.] http://di002.edv.uniovi.es/~cueva/investigacion/tesis/Fernando.pdf.

Cohn, Mike. 205.*Agile Estimating and Planing.* 205. ISBN 0-13-147941-5.

G Figueroa, Robert, A Cabrera, Armando y J Solís, Camilo. 2007.*Metodologías Tradicionales vs Metodologías Ágiles.* 2007.

Garlan, David y Shaw, Mary. 1903.*An introduction to software architecture. Advances in Software Engineering and Knowledge Engineering.* 1903. vol1.

github. 2017. distcc -- a free distributed C/C++ compiler system. [En línea] 2017. [Citado el: 29 de 11 de 2017.] https://github.com/distcc/distcc.

Gómez, Ramón M. Labrador. 2015. PROGRAMACIÓN AVANZADA EN SHELL. [En línea] 2015. [Citado el: 9 de 12 de 2017.] http://www.informatica.us.es/~ramon/articulos/Programacion-BASH.pdf.

Guerrero , Sonia Lambert, y otros. 2012. Estado actual de los sistemas de construcción de paquetes en diferentes distribuciones de GNU/Linux. [En línea] 4 de 09 de 2012. [Citado el: 22 de 11 de 2017.] http://rcci.uci.cu/?journal=rcci&page=article&op=view&path%5B%5D=198&path%5B%5D=161.

Hernández, Enrique Orallo. 2015. El Lenguaje Unificado de Modelado (UML) . [En línea] 2015. [Citado el: 6 de 12 de 2017.] http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

Larman, Craig. 2004.*UML y Patrones.* s.l. : Indiana, 2004.

Lic.Pérez, Dariem Herrera. 2013. Sistema distribuido para automatizar la construcción de repositorios de paquetes binarios de la distribución cubana de GNU/Linux Nova. [En línea] 2013. [Citado el: 28 de 1 de 2018.] http://dspace.uclv.edu.cu/bitstream/handle/123456789/7219/2013-01-31%20Tesis_Nova_Distributed_Build_System.pdf?sequence=1&isAllowed=y.

Maida, Esteban Gabriel. 2015. Metodologías de Desarrollo de Software . [En línea] diciembre de 2015. [Citado el: 5 de 12 de 2017.] http://bibliotecadigital.uca.edu.ar/repositorio/tesis/metodologias-desarrollo-software.pdf.

Microsoft. 2018. Developer Network. [En línea] 2018. [Citado el: 5 de 3 de 2018.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.

Montano, Ronny Martínez. 2016. Herramienta de sincronización entre los sistemas operativos GNU/Linux Nova y Android. [En línea] 2016. [Citado el: 14 de 12 de 2017.] https://repositorio_institucional.uci.cu/jspui/bitstream/123456789/7531/1/TD_08439_16.pdf.

Pérez, Dariem Herrera. 2013. Sistema distribuido para automatizar la construcción de repositorios de paquetes binarios de la distribución cubana de GNU/Linux Nova.Tesis de Maestría. [En línea] 2013. [Citado el: 25 de Octubre de 2017.] http://dspace.uclv.edu.cu/bitstream/handle/123456789/7219/2013-01-31%20Tesis_Nova_Distributed_Build_System.pdf?sequence=1&isAllowed=y.

Pérez, Lic. Dariem Herrera. 2013. Sistema distribuido para automatizar la construcción de repositorios de paquetes binarios de la distribución cubana de GNU/Linux Nova. [En línea] 2013. [Citado el: 20 de Noviembre de 2017.] http://dspace.uclv.edu.cu/bitstream/handle/123456789/7219/2013-01-31%20Tesis_Nova_Distributed_Build_System.pdf?sequence=1&isAllowed=y.

Pressman, Roger S. 2010. Ingeniería del software.Un enfoque práctico. [En línea] 2010. [Citado el: 25 de 2 de 2018.] http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF.

Quevedo, Ricardo Mejías y Pérez , Héctor Baranda. 2012. Sistema de compilación distribuida de Nova.Trabajo de Diploma. [En línea] junio de 2012. [Citado el: 25 de octubre de 2017.] https://repositorio.uci.cu/jspui/handle/ident/TD_05164_12.

Rodríguez, Tamara Sánchez. 2015.*Metodología de desarrollo para la Actividad productiva de la UCI.* 2015.

Sommerville. 2005. SlideShare. [En línea] 2005. [Citado el: 21 de 2 de 2018.] https://es.slideshare.net/jasc_584/ingenieriadesoftware-iansommerville7maedicion-9417118.

Villar, Andrés Monserrat. 2010. Soluciones distribuidas para una aplicación cliente/servidor de simulación.Tesis de Máster en Computación Paralela y Distribuida. [En línea] 2010. [Citado el: 1 de 3 de 2018.] https://riunet.upv.es/bitstream/handle/10251/13618/TESINA.PDF?sequence=1.

ANEXOS

Anexo 1: Entrevista para la captura de requisitos

Objetivo: Identificar los principales requisitos funcionales y no funcionales que debería satisfacer el proceso de compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux nova.

Preguntas:

1. ¿Qué es la compilación de paquetes?
2. ¿Cómo se realiza la compilación de paquetes en Nova?
3. ¿Cuáles son las principales deficiencias en la realización de este proceso actualmente?
4. ¿Qué herramientas de compilación se conocen para este proceso?
5. ¿Qué ventajas representa la utilización de una herramienta de compilación distribuida para Nova?
6. ¿Qué funcionalidades deberían estar presentes en la herramienta?

Anexo 2: Encuesta para evaluar el índice de satisfacción grupal de potenciales usuarios

Especialista, le invito a responder el siguiente cuestionario con el fin de conseguir su colaboración en la presente investigación, solicito que exprese en sus respuestas criterios verídicos que guíen al autor de la investigación. Marque con una “x” en una sola opción y en el caso de la pregunta 5 responda brevemente. Por el tiempo brindado, muchas gracias.

1. ¿Considera que la compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova es importante?

   Si ___No ___No sé___

2. ¿Cree usted que la forma en que se compilan actualmente los paquetes de código fuente durante el desarrollo de la distribución cubana GNU/Linux Nova es ágil?

   Si ___No ___No sé___

3. ¿Considera usted factible disponer de una herramienta para agilizar la compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova?

   Si ___No ___No sé___

4. Luego de haber visto la herramienta para agilizar la compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova refleje en qué medida le gusta la solución desarrollada.

   __Me gusta mucho
   __Me gusta más de lo que me disgusta
   __Me da lo mismo
   __Me disgusta más de lo que me gusta
   __No me gusta nada
   __No sé decir

5. ¿Qué opina usted acerca de los beneficios que traería para el Centro de Software Libre disponer de una herramienta que agilice la compilación distribuida de paquetes de código fuente en la distribución cubana GNU/Linux Nova?

Anexo 3: Acta de aceptación

Universidad de las Ciencias Informáticas

Acta de aceptación de productos de trabajo

ACTA DE ACEPTACIÓN DE PRODUCTOS DE TRABAJO

En cumplimiento del **Convenio de colaboración** establecido entre el **Centro de Software Libre (CESOL)** y el estudiante **Humberto Pérez Arbolaez** de la Facultad 1 de la Universidad de las Ciencias Informáticas y en función de la ejecución del proyecto: **Nova-compiler**, se hace entrega del producto que se relaciona a continuación:

- Herramienta de compilación distribuida de Nova (Nova-compiler)

La parte Cliente, luego de haber revisado el producto de trabajo relacionado anteriormente procede a firmar la aceptación de los mismos en total conformidad.

Entrega	Recibe
Nombre y apellidos:	**Nombre y apellidos:**
Humberto Pérez Arbolaez	Yoandy Pérez Villazón
Cargo: Estudiante Facultad 1	**Cargo:** Director de CESOL
Firma:	**Firma:**

Centro CESOL Facultad 1

Fecha: 10/05/2018