**Universidad de las Ciencias Informáticas**

**Facultad 3**


# Título: Instalador para la distribución GNU/Linux Nova de tipo kiosko.

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

**Autor(es):**

Miraida González Ponce

**Tutores:**

Ing. Javier Piñeiro Cárdenas

Ing. Arletis Velázquez Ramírez

**Junio, 2020, La Habana, Cuba**

**“Año 61 del Triunfo de la Revolución”**

# DECLARACIÓN JURADA DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firman la presente a los 30 días del mes de septiembre del año 2020.

_____

Miraida de la Caridad González Ponce

Autor

_____ _____

Ing. Arletis Velázquez Ramírez

Tutora

Javier Piñeiro Cárdenas

Tutor

## AGRADECIMIENTOS

Hoy es el día más importante de mi vida como estudiante, hoy he conquistado un gran sueño que nohubiese sido posible alcanzar sin el inmenso amor e incondicional apoyo de mi familia, la entrega deprofesores y la ayuda de amigos. Agradezco desde lo más profundo de mí:

A Dios por darme fuerzas para seguir adelante y acompañarme en los momentos más difíciles.

A mi madre que representa el mayor ejemplo de sacrificio que conozco, por sus consentimientosy por sus concejos certeros a lo largo de esta carrera.

A mi padre por apoyarme en el momento que más lo necesitaba parasalir adelante y por ayudarme a lograr uno de mis mayores sueños: ser una ingeniera.

A mis tutores Javier y Arletis por su guía certera.

A todos aquellos profesores que fueron incansables en los pasados cinco años.

A mis compañeros de aula.

A todas aquellas personas no mencionadas que de alguna forma u otra han contribuido a miformación y a los que no me ayudaron también.

## DEDICATORIA

A mis padres por las infinitas muestras de amor, sacrificio y entrega incondicional. Por guiarme en cada paso que he dado en la vida, por contribuir en mi educación y superación profesional.

## RESUMEN

La Universidad de las Ciencias Informáticas (UCI) tiene como reto fundamental el formar y preparar a jóvenes capacitados para informatizar la sociedad, reto que le ha correspondido de manera histórica. Cuenta con una facultad que estudia y desarrolla sistemas operativos basados en tecnologías libres, Nova distribución GNU/Linux es un ejemplo de ellos, creado con el propósito de brindar soberanía tecnológica al usuario y apoyar la migración a tecnologías de Software Libre y Código abierto como parte del proceso de Informatización en Cuba.

En esta investigación se presenta el proceso de desarrollo de una nueva versión del instalador para la distribución GNU/Linux Nova de tipo kiosko, como una solución independiente en las líneas de desarrollo de la distribución. El objetivo del presente trabajo de diploma es desarrollar una herramienta de instalación que permita agilizar y guiar al usuario en el proceso de instalación y la configuración del sistema.

Para guiar el proceso de desarrollo del sistema se utilizó la metodología de software Variación de AUP para la universidad y en la implementación fue utilizada la interfaz de líneas de comandos, se empleó el lenguaje Python y la librería Python dialog, Visual Paradigm fue utilizado como herramienta de modelado de ciclo completo. La validación de la aplicación se realizó mediante pruebas de software para garantizar agilidaden el proceso de instalación. Para realizar estas pruebas se aplicó el método de caja blanca y la técnica de camino básico. Este trabajo constituye un aporte al desarrollo de los instaladores de sistemas operativos GNU/Linux Nova.

*PALABRAS CLAVES:instalador, sistema operativo, kiosko, Nova*

## ABSTRACT

The University of the Information-Technology Sciences ( UCI ) has like challenge to base to form and training young people qualified to computerize the society, challenge that has concerned him of historic way. Nova counts on a faculty that you go into and that develops operating systems based in free technologies, distribution GNU/Linux is an example of them, created in order to offer the user technological sovereignty and to back up the migration to technologies of Freeware and Code opened as part of Computerization process in Cuba.

GNU/Linux encounters the process of development of a new version of the fitter for the distribution in this investigation Nova of type stand, like an independent solution in the lines of development of distribution. The objective of the present work of diploma is to unroll a tool of installation that you allow to speed up, to energize and to guide the user in the process of installation and the configuration of the system.

Variation of AUP utilized the methodology of software himself for the university in order to direct the process of development of the system and in implementation the interface of lines of commands was used, used him the language Python and the bookstore Python dialog, Visual Paradigm was used as tool of modeling of complete cycle. The validation of application came true by means of acceptance tests. The cash-basis method applied white note and the technique of basic road themselves in order to accomplish these proofs. This work constitutes a contribution to the development of the fitters of operating systems GNU/Linux Makes a Novation .


*KEYWORDS: Fitter, operating system, stand, Nova*

# CONTENIDO

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# INTRODUCCIÓN

Cuba tiene como uno de los principales desafíos informatizar la sociedad para contribuir al mejoramiento del desarrollo económico del país y, por ende, a la calidad de vida de la población. Se trabaja para eliminar la dependencia tecnológica. Para cumplir el objetivo se ha planteado la estrategia de migrar las plataformas, herramientas y servicios informáticos de todas las empresas e instituciones cubanas a tecnologías de software libre y código abierto[1].

Una institución líder en el proceso de migración es la Universidad de las Ciencias Informáticas (UCI), que cuenta con una facultad que estudia y desarrolla sistemas operativos basados en tecnologías libres, que permite a las personas operar sus computadoras sin restricciones de uso, desarrollo y mejoramiento.

En el mundo existen sistemas operativos libres como las muchas distribuciones de GNU/Linux entre las cuales se encuentra Nova, distribución desarrollada por estudiantes y profesores de la UCI, con la participación de miembros de otras instituciones para apoyar la migración a tecnologías de Software Libre y Código Abierto que experimenta Cuba como parte del proceso de Informatización de la Sociedad[2].

Nova provee un sistema cómodo, enfocado al usuario final, garantizando una interacción instuitiva que persigue minimizar el cambio brusco al que se enfrentan las personas familiarizadas con sistemas Microsoft Windows. Permite realizar trabajos de oficina, reproducir archivos de música y vídeo, navegar por Internet, ver fotografías y utilizar múltiples aplicaciones útiles para su desempeño laboral y momentos de ocio[2].

Entre los productos que se desarrollan para la distribución GNU/Linux Nova de tipo kiosko está su instalador, que tiene como función guiar al usuario en el proceso de instalación y configuración del sistema en el ordenador. A pesar de que el proceso actual de instalación cumple las condiciones básicas para instalar un sistema operativo, no se adapta a las necesidades del país. Se le llama kiosko porque no tiene un navegar de archivos.

Un kiosko es un software que brinda una interfaz mínima que permite ejecutar programas con un propósito determinado consumiendo la menor cantidad de recursos posibles[3].

Actualmente el proceso de instalación para la distribución GNU/Linux Nova de tipo kiosko se realiza desde una consola en modo texto, siendo un flujo de trabajo poco intuitivo y amigable para los usuarios. Esto es debido a que el cliente necesita poseer conocimientos técnicos de interacción con la Interfaz de Línea de Comandos (CLI). Por

tal motivo los usuarios con poco dominio para el trabajo con CLI pierden tiempo en el proceso de instalación de una personalización de la distribución GNU/Linux Nova tipo kiosko.

Por otra parte, los tamaños de las particiones que se crean durante este proceso están definidos de manera estática. Por lo que, para personalizar este paso del proceso de instalación, atendiendo a las características y necesidades propias de las instituciones se hace necesario realizar modificaciones al código fuente del instalador. Todo lo anterior, trae como consecuencia que los usuarios no puedan definir dinámicamente los tamaños de una partición durante el proceso de instalación de esta versión de Nova. Además, en estaciones de trabajo con una tabla de particiones del disco duro en formato GPT, es necesario su eliminación y la creación de una nueva tabla de particiones en formato MSDOS de forma manual desde la interfaz de línea de comandos antes de efectuar la instalación. Todo esto también provoca pérdida de tiempo en la instalación para la distribución GNU/Linux Nova de tipo kiosko.

La situación antes descrita conduce al siguiente **problema investigación**: ¿Cómo agilizar el proceso de instalación para la distribución GNU/Linux Nova de tipo kiosko?

Se plantea como **objeto de estudio**: Los instaladores de la distribución GNU/Linux. Delimitando como **campo de acción**: El instalador de la distribución GNU/Linux Nova.

Para dar solución al problema de investigación planteado se define como **objetivo general:** Desarrollar la versión 2.0 del instalador para la distribución GNU/Linux Nova de tipo kiosko, garantizando agilidaden el proceso de instalación.

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

1. Elaborar el marco teórico de la investigación sobre el instalador para la distribución GNU/Linux Nova de tipo kiosko.
2. Diseñar un instalador para la distribucción GNU/Linux Nova de tipo kiosko.
3. Implementar un instalador para la distribución GNU/Linux Nova de tipo kiosko.
4. Validar el objetivo de la investigación mediante la aplicación de técnicas y pruebas.

Se plantea como **hipótesis** en esta investigación:

Si se desarrolla la versión 2.0 del instalador para la distribución GNU/Linux Nova de tipo kiosko, se garantiza agilidad en el proceso de instalación.

En el desarrollo del trabajo de diploma se utilizaron los siguientes métodos de investigación

científica:

**Métodos Teóricos:**

- Inductivo_Deductivo: Se realiza para establecer una forma de razonamiento acerca de las particularidades de la herramienta de instalación para la distribución GNU/Linux Nova de tipo kiosko.
- La modelación: Permitió modelar la realidad de la solución mediante modelos y diagramas los diferentes elementos que la componen.

**Métodos Empíricos:**

- La encuesta: Se realiza para obtener información sobre las deficiencias existentes del instalador para la distribución GNU/Linux Nova de tipo kiosko y los elementos a tener en cuenta en el desarrollo de la solución.

La entrevista: Se realizó a los especialistas del centro de software libre CESOL para obtener toda la información del proceso del instalador para la distribución GNU/Linux Nova de tipokiosko.

**Estructura del trabajo de diploma:**

El presente documento está estructurado en: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos.

**Capítulo1:** Fundamentación teórica sobre el proceso de instalación para la distribución GNU/Linux Nova de tipo kiosko.

En este capítulo se abordará los principales conceptos en la investigación, se realiza un estudio sobre la herramienta de instalación para la distribución GNU/Linux Nova de tipo kiosko, además se define la metodología de desarrollo de software a emplear y propone los lenguajes de programación, las herramientas y tecnologías a utilizar en el desarrollo de la propuesta de solución.

**Capítulo 2:** Análisis y diseño del instalador para la distribución GNU/Linux Nova de tipo kiosko.

En el capítulo se presenta la propuesta de solución al problema de la investigación. Se obtiene los productos de trabajo resultantes del desarrollo de las actividades ejecutadas en las disciplinas de Modelado del negocio, Requisitos, Análisis y diseño e implementación que se definen en la metodología.

**Capítulo 3:** Implementación, pruebas y evaluación del instalador para la distribución GNU/Linux Nova de tipo kiosko.

El capítulo contiene los elementos que forman parte de la validación de la solución. En él se exponen las pruebas realizadas a la solución para validar las funcionalidades del sistema desarrollado, así como los resultados obtenidos en la evaluación del objetivo de la investigación.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA SOBRE EL PROCESO DE INSTALACIÓN PARA LA DISTRIBUCIÓN GNU/LINUX NOVA DE TIPO KIOSKO.

## 1.1. Introducción

En el presente capítulo se muestra la definición de conceptos para un mejor entendimiento del tema de investigación, se realiza la fundamentación de la propuesta de solución y se determina los sistemas homólogos. Se define la metodología a utilizar para guiar el proceso de desarrollo de software y se determinaron los lenguajes y herramientas que se emplean para la realización del trabajo.

## 1.2. Definición de conceptos

**Instalador:** Es un paquete que contiene código que está comprimido y se puede instalar en un computador dependiendo el sistema operativo y la plataforma a la que se desea instalar o sea enraizar el código en la máquina para pasar a ser ejecutado o inicializado[5].

**Sistema Operativo:** Conjunto de órdenes y programas que controlan los procesos básicos de una computadora y permiten el funcionamiento de otros programas. Es una plataforma que facilita la interacción entre el usuario, los demás programas del ordenador y los dispositivos de hardware[6].

**Python:**Es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes, de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas. El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C), también puede usarse como un lenguaje de extensiones para aplicaciones personalizables[7].

**Interfaz de Línea de Comando (CLI):** Es un utilitario de configuración basado en texto que admita un conjunto de comandos y parámetros del teclado paraconfigurar y gestionar un sistema[8]. Interfaz gráfica: Es un programa informático formado por imágenes y objetos gráficos, que representan la información y acciones que se encuentran en la interfaz. Su objetivo es el de crear un entorno visual fácil de usar para que fluya la comunicación son el sistema operativo [9].

**Formato GTP:** Esun estándar de particionado para la colocación de la tabla de particiones enun disco duro físico. Está orientado a ser el sistema propio para la Interfaz de firmware extensible unificada (UEFI), implementada para los modernos sistemas Extensible Firmware Interface[10].

**Interfaz de firmware extensible unificada (UEFI):** Es el código del firmware de un chip en la placa base que proporciona funciones adicionales a las del sistema de entrada/salida básico (BIOS). UEFI ofrece una manera de hacer cosas con el equipo antes de que se cargue un sistema operativo[11].

**Formato MSDOS** (siglas de **M**icro**S**oft **D**isk **O**perating **S**ystem, Sistema operativo dedisco de Microsoft) es sustituido gradualmente por sistemas operativos que ofrecen una interfaz gráfica de usuario, en particular por varias generaciones de Microsoft Windows[12].

## 1.3. Análisis de sistemas homólogos.

**YaSt:** Es la herramienta de instalación y configuración de open SUSE y la distribución SUSE Linux Enterprise. Es popular por su facilidad de uso y la interfaz gráfica atractiva y la capacidad de personalizar su sistema de forma rápida durante y después de la instalación. YaST es un acrónimo del inglés **Y**et **a**nother **S**etup **T**ool lo que se podría traducir como: otra herramienta de configuración. YaST se puede utilizar para configurar todo el sistema. La configuración de hardware, configurar la red, los servicios del sistema y ajustar la configuración de seguridad. A todas estas tareas se puede llegar desde el centro de control YaST[13].

**Ubiquity:** Es un simple instalador gráfico de CD en vivo diseñado para integrarse bien con los sistemas basados en Debian y Ubuntu. Es mantenido por el equipo instalador de Ubuntu. Se ejecuta desde el Live CD o USB. Ubiquity cuenta con diferentes interfaces para cada uno de sus productos (Ubuntu, Kubuntu, Edubuntu, Xubuntu), dependiendo del entorno de escritorio que estos utilicen por defecto (GNOME, KDE). Todas las interfaces están implementadas utilizando el lenguaje de programación Python, pero utilizan librerías gráficas diferentes, GTK para los productos que utilizan GNOME y QT para los que usan KDE[14].

**Debian-Installer:** Es el sistema de software para instalar un sistema básico de Debian que funcione. Se admite una amplia gama de hardware, como dispositivos integrados, computadoras portátiles, computadoras de escritorio y máquinas de servidor. La instalación se realiza respondiendo un conjunto básico de preguntas. También está disponible un modo experto que permite controlar cada aspecto de la instalación y una función avanzada

para realizarinstalaciones automatizadas. El sistema instalado se puede usar tal cual o personalizar aún más. La instalación se puede realizar desde múltiples fuentes: USB, CD / DVD / red. El instalador admite instalaciones localizadas en más de 80 idiomas[15].

**Serere:** Es la encargada de instalar el sistema en el disco duro del ordenador. Puede usarse de dos maneras; en modo automático y manual, con un tiempo de instalación bastante corto y sencillo. Se busca expandir su funcionamiento para que Nova pueda ser instalada en la red y dispositivos de almacenamiento externo, como también adherirse al gestor de paquetes de Nova[2].

### 1.3.1 Comparación entre los sistemas homólogos.

**Tabla 1.Tabla de los sistemas homólogos**

| Sistemas homólogos | Tipo de Licencia (Privada) | Instalador Gráfico | Plataformas en las que está disponible Windows | Consume pocos recursos | Poca Documentación | Código fuente no disponible | Instalación lenta | Ámbito nacional | Particionado Dinámico |
|---|---|---|---|---|---|---|---|---|---|
| **YaST** | No | Si | No | No | No | No | No | No | Si |
| **Ubiquity** | No | Si | No | No | No | No | No | No | Si |
| **Debian-Installer** | No | Si | No | Si | No | No | Si | No | Si |
| **Serere** | Si | No | No | Si | Si | Si | No | Si | No |

Después de haber realizado un estudio de los indicadores previamente seleccionados se concluye lo siguiente:

EL instalador YaST y el instalador Ubiquity han sido descartados por su diferencia de enfoque en algunas funcionalidades y características que no son factibles para la situación de nuestro país. Por ejemplo, aquellas aplicaciones instaladoras que consumen demasiados recursos, no incluyen interfaces ligeras de instalación, o no interactúan con hardware obsoleto.

El Debian-Installer ha sido descartado porque su instalación es difícil para un usuario sin conocimiento de Linux y el instalador Serere se descartó porque no tiene documentación abundante para el proceso de instalación y su código fuente no está disponible.

A partir del estudio realizado y conociendo el funcionamiento de algunos de los instaladores, de las distribuciones GNU/Linux más usadas en el mundo, es posible afirmar que la versión actual del instalador de Nova de tipo kiosko, no cuenta con varias de las características y funcionalidades más importantes para este tipo de aplicación.

Con el propósito de mejorar el proceso de instalación de Nova, debe considerarse incluir entre las funcionalidades de la versión 2.0 del instalador para la distribución GNU/Linux Nova de tipo kiosko, la posibilidad de configurar el usuario y contraseña. Además, debe permitir particionar dinámicamente. También es necesario diseñar una interfaz amigable, fácil de utilizar y donde el usuario interactúe en un ambiente simple durante todo el proceso de instalación.

Debe tenerse en cuenta la posibilidad reutilización de código y componentes de las versiones anteriores del instalador y de otros instaladores que se ajusten a las condiciones de desarrollo de Serere 2.0.

## 1.4. Metodología de desarrollo de software.

Una metodología de desarrollo de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. La metodología que se utiliza es la variación AUP para la UCI ya que esta metodología logra estandarizar el proceso de desarrollo de software en los proyectos productivos de la UCI. Las características de las fases de la metodología son[16]:

1. **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 del cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
2. **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, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
3. **Cierre:** En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

**Disciplinas de Variación AUP para la UCI.**

AUP propone de 7 disciplinas[16]:

- **Modelado de negocio:**Es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Para modelar el negocio se proponen las siguientes variantes:

1- Casos de Uso del Negocio (CUN).
2- Descripción de Proceso de Negocio (DPN).
3- Modelo Conceptual (MC). A partir de las variantes anteriores se condicionan cuatro escenarios para modelar el sistema en la disciplina Requisitos.

En el modelado del desarrollo del instalador para la distribución GNU/Linux Nova de tipo kiosko se utilizó la variante de modelo conceptual.

- Requisitos:El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. Existen tres formas de encapsular losrequisitos [Casos de Uso del Sistema (CUS), Historias de usuario (HU) y Descripción de requisitos por proceso (DRP)].

  Los requisitos se encapsulan en Historias de Usuarios para el desarrollo de la herramienta de instalación.

- **Análisis y diseño:** En esta disciplina, si se considera necesario, los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura). Además, en esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales. Los modelos desarrollados son más formales y específicos que el de análisis.

  Para el desarrollo del instalador para GNU/Linux Nova de tipo kiosko se realizó el diagrama de paquete que ayuda a la estructuración del sistema.

- **Implementación:** En la implementación, a partir de los resultados del Análisis y Diseño se construye el sistema.

- **Pruebas internas:** En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posible componentes de prueba ejecutables para automatizar las pruebas.

- **Pruebas de Aceptación:** Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales

para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

**Escenario para la disciplina de requisitos[16]:**

**Escenario No 4:** Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información.

## 1.5. Lenguajes y herramientas para el modelado de solución.

En el modelado de la propuesta de solución se utilizó el Lenguaje Unificado de Modelado y la herramienta Visual Paradigm descritos a continuación:

### 1.5.1 Lenguaje de modelado UML

El lenguaje de modelado unificado (UML) es un estándar para la representación visual de objetos, estados y procesos dentro de un sistema. Por un lado, el lenguaje de modelado puede servir de modelo para un proyecto y garantizar así una arquitectura de información estructurada; por el otro, ayuda a los desarrolladores a presentar la descripción del sistema de una manera que sea comprensible para quienes están fuera del campo. UML se utiliza principalmente en el desarrollo de software orientado a objetos. Al ampliar el estándar en la versión 2.0, también es adecuado para visualizar procesos empresariales[17]. Este lenguaje es utilizado en varias herramientas CASE (Computer Aided Software Engineering) con el objetivo de permitir la productividad y el control de la calidad en cualquier proceso de elaboración del software.

### 1.5.2 Herramienta de modelado Visual Paradigm

Visual Paradigm es una herramienta UML CASE: Ingeniería de Software Asistida por Computación, que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue, permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación[18].

Para la modelación de cada uno de los procesos y artefactos en la presente investigación, se utilizará la herramienta Visual Paradigm en su versión 8.0, es una herramienta de fácil aprendizaje, fácil de emplear y con abundante documentación.

## 1.6. Tecnologías de implementación.

**Lenguaje Python:** Es un lenguaje de programación que se destaca por su código legible

y limpio. Una de las razones de su éxito es que cuenta con una licencia de código abierto que permite su utilización en cualquier escenario. Esto hace que sea uno de los lenguajes de iniciación. Esun lenguaje sencillo, legible y elegante que atiende a un conjunto de reglas que hacen muy corta su curva de aprendizaje[7].

**Pythondialog:** Es un contenedor de Python para la utilidad de diálogo originalmente escrita por Savio Lam, y luego reescrita por Thomas E. Dickey. Su propósito es proporcionar una interfaz Python fácil de usar, pitón y completa para el diálogo. Esto permite crear interfaces de usuario simples en modo de texto en sistemas tipo Unix (incluido Linux)[19].

## 1.7. Conclusiones del Capítulo.

En este capítulo se mostró una visión teórica de los principales conceptos que van a estar presentes durante todo el ciclo del trabajo de diploma, se determinó a partir de los sistemas homólogos la necesaria modificación de la herramienta de instalación para la distribución GNU/Linux Nova de tipo kiosko que permita agilizar dicho proceso, así como las tecnologías y herramientas a utilizar en la misma para dar cumplimiento a los objetivos propuestos y así poder crear un producto amigable y de calidad para el cliente. Como lenguaje de programación se escogió Python para el desarrollo de dicho instalador y se utilizará la librería de Pythondialog para realizar la interfaz gráfica. Se definió la metodología AUP para la UCI con sus fases, disciplinas y escenarios para guiar el proceso de desarrollo. Se utilizará la herramienta Visual Paradigm para modelar la herramienta de instalación.

# CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL INSTALADOR PARA LA DISTRIBUCIÓN GNU/LINUX NOVA DE TIPO KIOSKO.

## 2.1. Introducción.

En este capítulo se define el diseño de la aplicación, los estándares y patrones que se utiliza y la arquitectura de acuerdo a la metodología empleada. Se describen las características que posee el instalador para GNU/Linux Nova de tipo kiosko y se identifican los requisitos funcionales y no funcionales. Para construir un entorno que posibilite lo antes planteado se tuvieron en cuenta las herramientas ya descritas para el modelado.

## 2.2. Propuesta de solución.

Con el objetivo de resolver el problema de investigación planteado en el capítulo anterior, se propone modificar el instalador para la distribución GNU/Linux Nova de tipo kiosko. Teniendo como meta diseñar una interfaz amigable, fácil de utilizar y donde el usuario interactúe en un ambiente simple durante todo el proceso de instalación. Con la nueva versión del instalador se establecen perfiles de particionado, lo que permitirá ajustar la instalación a los requerimientos de almacenamiento a cada una de las instituciones y computadoras. Además con las nuevas modificaciones el instalador sea capaz de detectar y cambiar a formato MSDOS las tablas de particiones en GPT permitiendo la instalación satisfactoria en estaciones de trabajo que así lo requieran.

## 2.3. Modelado del negocio.

La modelación del negocio consiste en comprender los procesos de negocio de una organización que se desean informatizar, con el objetivo de garantizar que el software desarrollado va a responder a las necesidades del cliente. En este proceso el equipo de desarrollo de software debe tener un contacto directo con el cliente y se puede auxiliar de su documentación[20].

Para comprender el contexto del negocio a informatizar en el desarrollo del instalador para la distribución GNU/Linux Nova de tipo kiosko, la autora de la tesis elaboró un modelo conceptual, en el que plasmó los conceptos, sus características y las relaciones existentes entre ellos.

**Modelo Conceptual**

El Modelo conceptual es una representación visual de los objetos y conceptos (o partes esenciales) que se requieren en la modelación de un problema determinado y las relaciones que subsisten entre ellos[21]. En la ilustración 1 se muestra el Modelo conceptual que representa los conceptos del contexto del negocio referente al instalador para GNU/Linux de tipo kiosko.

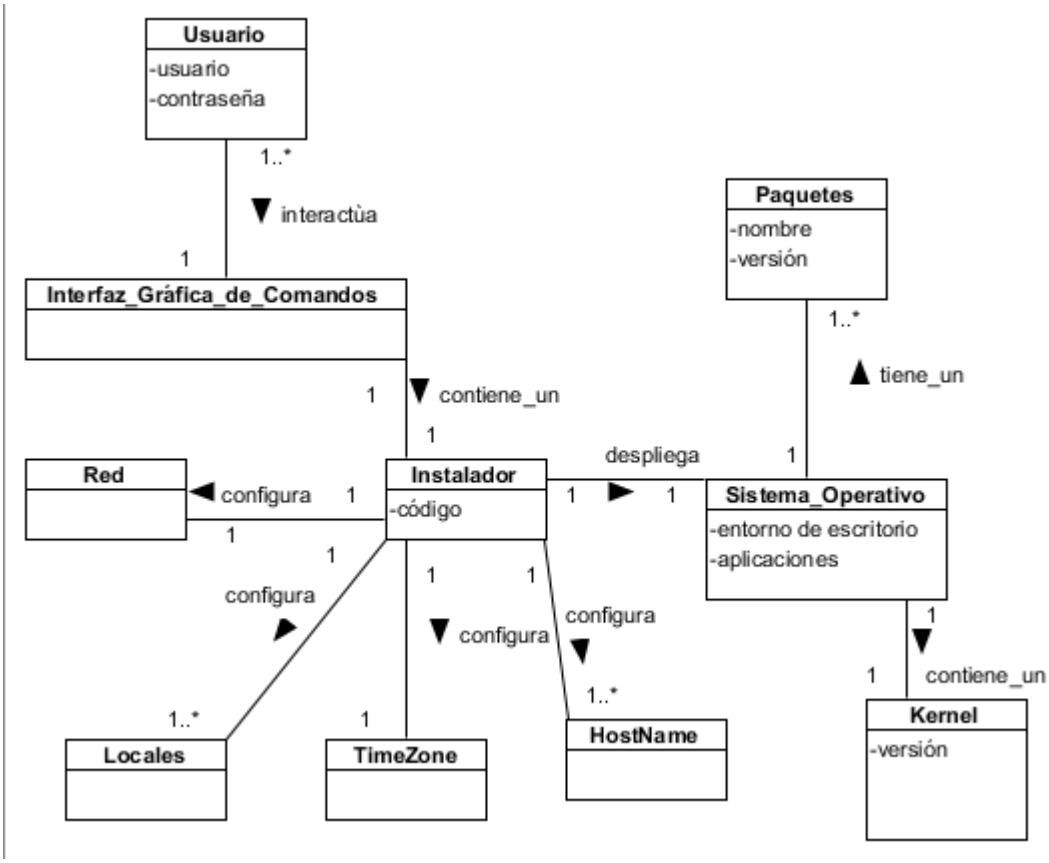



**Ilustración 1. Modelo Conceptual**

Los conceptos del contexto del negocio anteriormente modelado son los siguientes:

**Usuario:** contiene la información general de los usuarios.

**Interfaz Gráfica de Comandos:** donde el usuario se autentica.

**Instalador:** herramienta de instalación.

**Sistema Operativo:** donde se despliega el instalador.

**Paquetes:**es una colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.

**Kernel:** es el núcleo, la base del sistema operativo.

**Red:** configuración de la red del sistema

**Locales:** configuraciónde los IP del sistema

**TimeZone:** configuración de la zona horaria del sistema.

**HostName:** configuración del host del sistema.

Para comprender el contexto del negocio se elaboró el modelo conceptual, en el que se definen los conceptos asociados con la herramienta de instalación para la distribución GNU/Linux Nova de tipo kiosko, así como las características de estos conceptos y las relaciones existentes entre ellos. El modelo conceptual elaborado se utiliza como base para la definición de las funcionalidades y restricciones que debe cumplir el sistema.

## 2.4. Especificación de requisitos.

La especificación de requisitos de software (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software. Los casos de uso también son conocidos como requisitos funcionales. Además de los casos de uso, la ERS también contiene requisitos no funcionales[22].

### 2.4.1. Fuentes para la obtención de requisitos.

Las fuentes de obtención de requisitos utilizadas fueron:

- Modelo Conceptual
- Especialistas del Centro CESOL
- Análisis de las herramientas existentes

### 2.4.2. Técnicas de identificación de requisitos.

Las técnicas de identificación de requisitos de software permiten identificar las necesidades de negocio de clientes y usuarios. Son mecanismos que se utilizan para recolectar la información necesaria en la obtención de los requisitos funcionales y no funcionales de una aplicación[20].

En la identificación de los requisitos del sistema se utilizó:

- **Entrevista:** Se realizó una entrevista a especialistas del centro CESOL que permitió definir las características de la problemática existente, las posibles funcionalidades a tener en cuenta en el desarrollo de la solución y los beneficios que aportaría el sistema propuesto.

- **Desarrollo de los prototipos:** Permitió que el usuario tuviera una visión general de lo que desea y que puedan mejorar las especificaciones de los requerimientos.

### 2.4.3. Requisitos Funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. A partir del estudio y las investigaciones realizadas se obtuvieron los siguientes requisitos funcionales que debe cumplir el instalador con vista satisfacción final del cliente:

**Tabla 2. Listado de los requisitos funcionales de la propuesta de solución**

| RF | Nombre | Complejidad |
|---|---|---|
| RF.1 | Configurar el usuario | Media |
| RF.2 | Mostrar discos duros | Media |
| RF.3 | Eliminar tabla de particiones | Media |
| RF.4 | Crear tabla de particiones | Media |
| RF.5 | Instalar sistema operativo | Alta |

**Requisitos no funcionales:**

Los requisitos no funcionales son restricciones de las funciones o servicios ofrecidos por el sistema. No se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este. Los requisitos no funcionales definidos para el desarrollo de la aplicación propuesta se listan a continuación:

**Tabla 3.Listado de los requisitos no funcionales de la propuesta de solución.**

| No | Requisito | Sub-atributo | Descripción |
|---|---|---|---|
| RNF.1 | Usabilidad | Aprendibilidad | El instalador debe de ser fácil de usar y la interacción del usuario con sus funcionalidades debe lograrse de forma |

| | | | |
|---|---|---|---|
| | | | sencilla.<br>. |
| RNF.2 | Portabilidad | Adaptabilidad | El instalador funcionará sobre el sistema operativo GNU/Linux con la distribución Nova. |
| RNF.3 | Funcionalidad | Preciso | Se debe contar con un tiempo de respuesta rápido para la cuál es necesario que el instalador sea lo más estable y confiable posible. |
| RNF.4 | Confiabilidad | Madurez | El instalador deberá tener la capacidad de funcionar sin que se produzcan errores o con el mínimo posible que no afecte a su correcto funcionamiento. De esta forma los usuarios podrán tener confianza sobre el servicio que proporciona. |

### 2.4.5. Historias de Usuario.

Las historias de usuario son descripciones, simple, muy cortas y esquemáticas, que resumen la necesidad concreta de un usuario al utilizar el sistema, así como la solución que la satisface. Su función principal es identificar problemas percibidos, proponer soluciones y estimar el esfuerzo que requieren implementar las ideas propuestas[23]. En el proceso de desarrollo se realizó las siguientes historias de usuario:

**Tabla 4. Historia de Usuario 1**

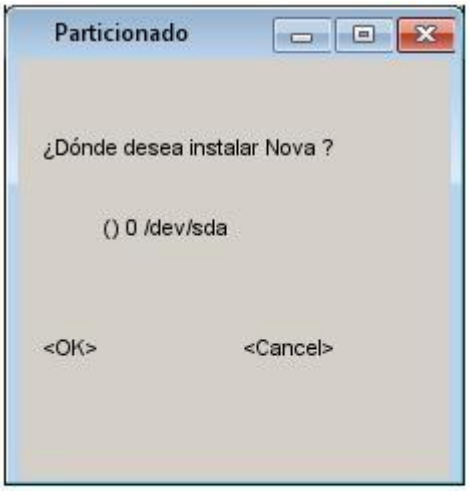
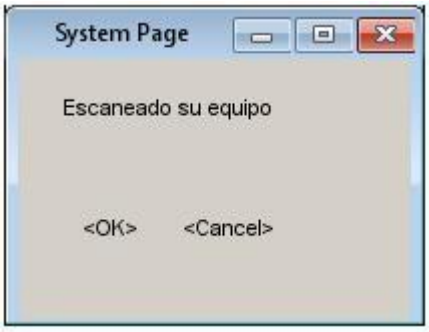
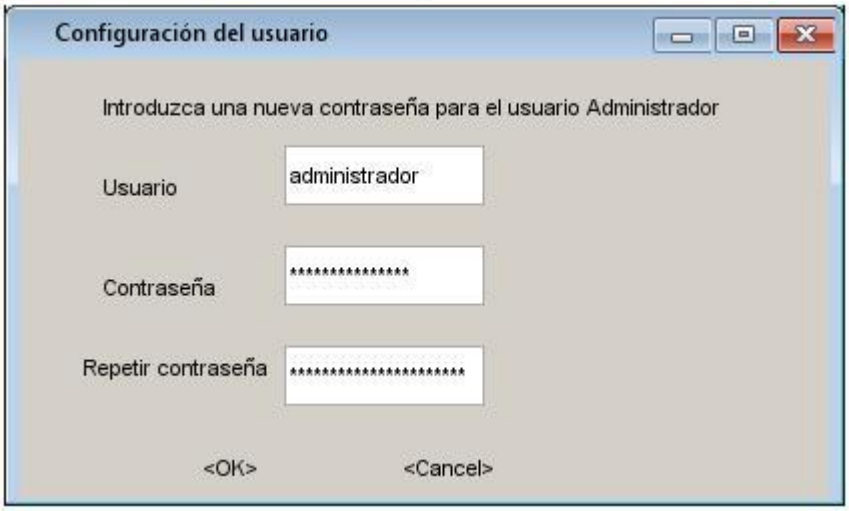
| | |
|---|---|
| **Número:** HU2 | **Requisito:** Mostrar discos duros |
| **Programador: Miraida** González Ponce | **Iteración Asignada:** 1 |
| **Prioridad**: Media | **Tiempo Estimado:** 72 horas |
| **Riesgo en Desarrollo**: Ausencia del desarrollador por enfermedad o pérdida de información imprescindible . | **Tiempo Real:** 52 horas |
| **Descripción:** El usuario selecciona el disco duro en que se realizará la instalación. | |
| **Observaciones: :** Si el usuario no selecciona el disco duro para el proceso de instalación le mostrará una notificación sobre este problema. | |
| **Prototipos de interfaz de usuario:**   | |

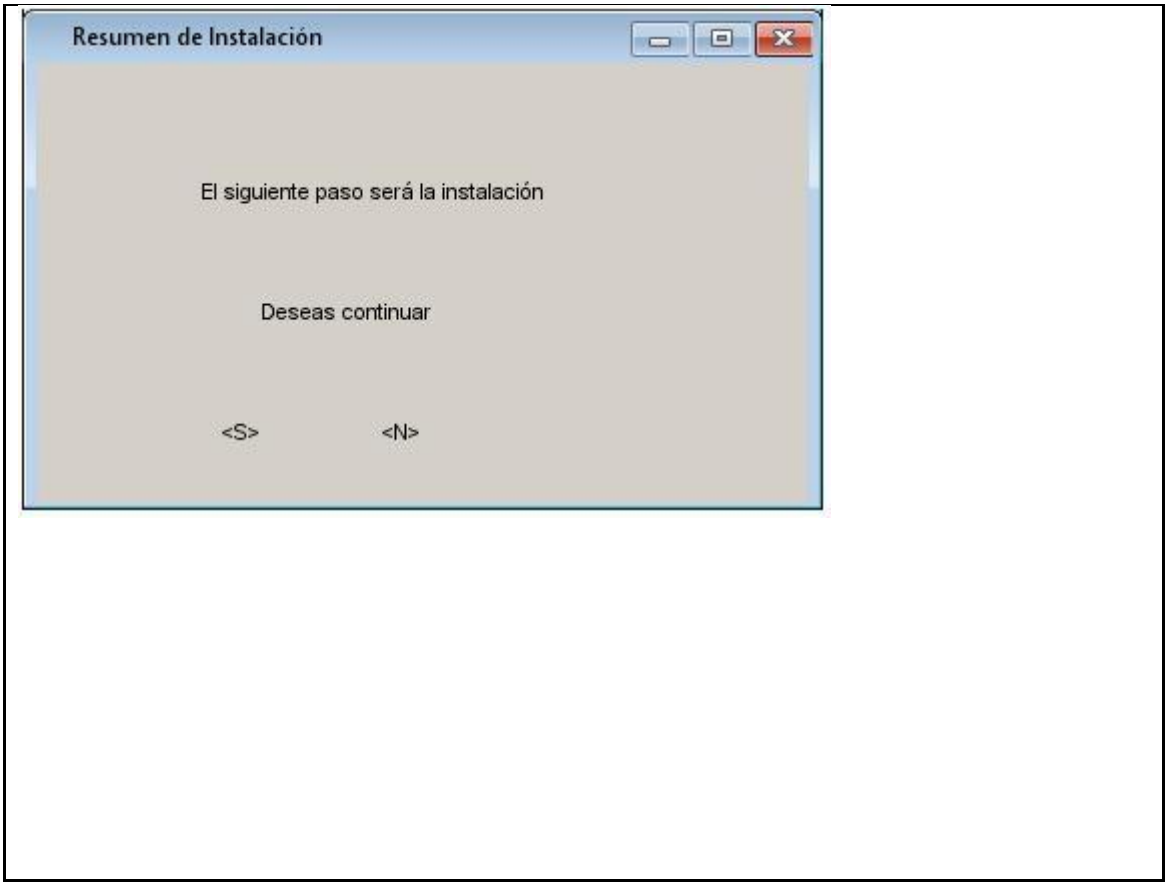
**Tabla 5. Historia de Usuario 2.**

| | |
|---|---|
| **Número:** HU5 | **Requisito:** Instalar el sistema operativo |

| | |
|---|---|
| **Programador:** Miraida González Ponce | **Iteración Asignada:** 1 |
| **Prioridad**: Alta | **Tiempo Estimado:** 72 horas |
| **Riesgo en Desarrollo**: Ausencia del desarrollador por enfermedad o pérdida de información imprescindible . | **Tiempo Real:** 52 horas |
| **Descripción:** El usuario realiza el proceso de instalación. | |
| **Observaciones:** Si el usuario no efectúa correctamente todos los paso para el proceso de instalación el sistema le mostrará una notificación sobre este problema. | |






**2.4.6. Validación de los requisitos de software.**

El objetivo principal de la actividad validar los requisitos del sistema es comprobar que el sistema software descrito por la Especificación de Requisitos del Sistema (ERS) se corresponde con las necesidades de negocio de clientes y usuarios, obteniendo su aprobación. Existen diferentes técnicas para la validación de requisitos de software como: Técnica de prototipado, Diseño de casos de prueba y Revisiones Técnicas Formales[22]:

- Técnica de prototipado: es una representación limitada de un producto, permite a las partes probarlo en situaciones reales o explorar su uso, creando así un proceso de diseño de iteración que genera calidad. Diseño de casos de pruebas: el objetivo principal del diseño de casos de prueba es obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software.
- Revisiones Técnicas Formales: es el filtro más efectivo desde el punto de vista de garantía de calidad. El objetivo fundamental de estas revisiones es encontrar errores durante el proceso de desarrollo de software. Se deben realizar desde el principio del proyecto para evitar la propagación de los errores a las etapas siguientes del proceso de software.

La validación de los requisitos de software de la propuesta de solución se desarrolló a través de las técnicas de prototipado y diseño de casos de prueba. En el proceso de validación de los requisitos se diseñaron los prototipos de interfaces. Esto permitió que el cliente tuviera una noción de la propuesta de solución y se pudieran hacer las correcciones necesarias antes del diseño e implementación de las funcionalidades del sistema. La generación de casos de prueba permitió especificar los posibles datos de entrada, así como las salidas del sistema generándose un diseño de caso de prueba para cada requisito funcional.

**Casos de Pruebas:**

Se diseñó los casos de pruebas siguientes para la herramienta de instalación:

**Tabla 6. Caso de Prueba.**

| Escenario | Descripción | Disco | Contraseña | Respuesta del sistema | Flujo Central |
|---|---|---|---|---|---|
| EC 5.1 Instalar sistema operativo correctamente. | El sistema debe permitir seleccionar si deseas comenzar con la instalación. | /dev/sda | ********** | 1.1 El sistema mostrará un mensaje: "Se ha instalado satisfactoriamente" 1.2 EL usuario selecciona la opción ok | 1.1 El usuario selecciona la opción ok. 1.2 El sistema comienza la instalación 1.3 El sistema muestra un mensaje: "se ha realizado la instalación correctamente" conecta al servidor para verificar que no existe una 1.4 EL sistema permite seleccionar al usuario la opción de reiniciar el sistema. |

| | | | | | |
|---|---|---|---|---|---|
| EC 5.2 Seleccionar el particionado del disco correctamente | El sistema debe permitir seleccionar el disco duro donde se realizará la partición. | /dev/sda | ********** | 1.1 El usuario selecciona el disco donde se realizará la partición.<br><br>1.2 Si el usuario no selecciona ningún disco, el sistema mostrará un mensaje: "Seleccionar disco"<br><br>1.3 El usuario selecciona la opción ok. | 1.1 El sistema valida que el usuario ha seleccionado un disco duro para el particionado. |

## 2.5. Análisis y diseño.

### Persistencia de datos

La persistencia de datos es la acción de preservar la información de un objeto de forma permanente (guardado), pero a su vez también se refiere a poder recuperar la información del mismo (leerlo) para que pueda ser nuevamente utilizado.

En las siguientes ilustraciones (2,3,4,5) se muestra la persistencia de datos del instalador para la distribución GNU/Linux Nova de tipo kiosko. Contiene ficheros donde, cada uno de los ficheros guarda la información para la configuración del sistema operativo.

En la ilustración 2 se observa cómo se abre el fichero timezone y se coloca en modo escritura, luego se escribe la nueva información y se cierra el fichero.

```
etc_timezone = open(os.path.join(DataHolder.INSTALL_DIR, "etc/timezone"), 'w')
etc_timezone.write(str(timezone) + '\n')
etc_timezone.close()
```

**Ilustración 2. Datos del fichero timezone**

En la ilustración se observa cómo se abre el fichero hostsPath y se coloca en modo escritura, luego se escribe la nueva información y se cierra el fichero.

```
file1 = open(hostsPath, "w")
file1.write("127.0.0.1 localhost")
file1.close()
```

**Ilustración 3. Datos del fichero hostsPath**

En la ilustración 4 se observa cómo se abren los ficheros locale y environmet, se coloca en modo escritura, luego se escribe la nueva información y se cierra el fichero.

```
file = open("%s/etc/default/locale" % (DataHolder.INSTALL_DIR), "w")
file.write("LANG='%s'\n"%lang)
file.close()

file = open("%s/etc/environment" % (DataHolder.INSTALL_DIR), "a")
file.write("LANG='%s'\n"%lang)
file.close()

file = open("%s/var/lib/locales/supported.d/local" % (DataHolder.INSTALL_DIR), "w")
file.write("%s UTF-8\n"%lang)
file.close()
```

**Ilustración 4. Datos de los ficheros locale y environment**

En la ilustración5 se observa cómo se abren los ficheros interfaces y 70-persistente-net.rules, se coloca en modo escritura, luego se escribe la nueva información y se cierra el fichero.

```
file = open("%s/etc/network/interfaces" % DataHolder.INSTALL_DIR, 'w')
file.write("auto lo\n")
file.close()

file = open("%s/etc/network/interfaces" % DataHolder.INSTALL_DIR, 'a')
file.write(valor)
file.close()

file = open("%s/etc/udev/rules.d/70-persistent-net.rules" % DataHolder.INSTALL_DIR, 'w')
file.write(regla)
file.close()
```

**Ilustración 5. Datos de los ficheros: interfaces y 70-persistent-net.rules**

**Perfiles de particionado para ajustar el proceso de instalación**

```
sizeSwap =  self.augeasMGR.getSerereValue("SIZE_Swap")
sizeRaiz =  self.augeasMGR.getSerereValue("SIZE_RAIZ")
```

**Ilustración 6. Perfiles de particionado establecidos**

```
self.__hdmanager.commitChanges(self.__device, 1, sizeRam, sizeRaiz)
self.__hdmanager.setupMountPoints(self.__device, imagePath)
```

**Ilustración 7. Particionando perfiles establecidos**

Con la nueva versión del instalador se establecen perfiles de particionado, lo que permitirá ajustar la instalación a los requerimientos de almacenamiento a cada una de las instituciones y computadoras. Anteriormente los valores de la Swap y la Raíz estaban en el código, actualmente se lee desde un fichero de configuración (ver ilustración6). En la ilustración7 se observa como se le pasan esos valores al hdManager que es el encargado de particionar.

**Tablas de particiones en el formato MSDOS**

```
if device.getDiskLabel() == "gpt":

    device.setDiskLabelToMSDOS()
```

**Ilustración 8. Verificación del formato de las tablas**

```
def setDiskLabelToMSDOS(self,):

    self._disk = parted.freshDisk(self._dev, "msdos")
    self._disklabel = self._disk.type
    self.update()
```

**Ilustración 9. Cambio de formato(MSDOS)**

Con las nuevas modificaciones el instalador es capaz de detectar y cambiar a formato MSDOS las tablas de particiones en GPT permitiendo la instalación satisfactoria en estaciones de trabajo que así lo requieran. Anteriormente no se verificaba el tipo de tabla de partición, resultando una instalación fallida cuando este era en formato GTP, con la nueva versión se verifica el tipo de formato, (ver en la ilustración8) si es en formato GTP cambia a formato MSDOS (ver en la ilustración9).

## 2.6. Modelado del diseño.

En el diseño de un sistema, los patrones de diseños son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de software, permite estandarizar el código, haciendo que el diseño sea más comprensible para otros programadores. Además, acorta la fase de diseño en un proceso de desarrollo de software. Uno de los más conocidos son los Patrones Generales de Software para Asignar Responsabilidades (GRASP, por sus siglas en inglés), que describen los principios fundamentales de la asignación de responsabilidades a objetos[23].

Se hace uso de los patrones de diseño GRASP porque aportan al código un alto grado de organización y proporcionan ayuda al programador en la implementación del software. Además, representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación, se describen los patrones que se utilizaron en el desarrollo del instalador para GNU/Linux Nova de tipo kiosko:

- Experto: Se basa en asignar una responsabilidad al experto en información. Un experto en información es una clase que cuenta con la información necesaria para cumplir con una responsabilidad indicada. De esta manera se garantiza la obtención de un diseño con mayor cohesión en su estructura, robusto, fácil de mantener y con la información centralizada en una misma clase[23]. En el desarrollo del instalador para GNU/Linux Nova de tipo kiosko se evidencia este patrón en la clase installerThread que es la encargada de realizar la instalación.
- Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos. Se aplicará en todos los casos donde una clase tiene la responsabilidad de crear una nueva instancia de la otra[23]. El uso de este patrón se evidencia en la clase userManager, ella es la encargada de crear instancia a la clase user[23].
- Bajo Acoplamiento: Consiste en tener las clases lo menos relacionadas posible. De forma tal que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases. Este patrón potencia la reutilización y disminuye la dependencia entre las clases[23]. En la implementación del instalador para GNU/Linux Nova de tipo kiosko se emplea este patrón en la clase hdManager donde solo está relacionada con la clase disk.

Un diseño bien estructurado garantiza la eficiencia y estabilidad de una aplicación. Este se rige por una serie de elementos como los diagramas, los que garantizan mejor organización y entendimiento a la hora de desarrollar un producto.

**Diagrama de clases**

Se realizó el diagrama de clases para describir la estructura del instalador para GNU/Linux Nova de tipo kiosko, en él se representan las clases junto a sus atributos y métodos, así como las relaciones entre ellas. Se realiza en el proceso de análisis y diseño, el cual se define de la siguiente forma:

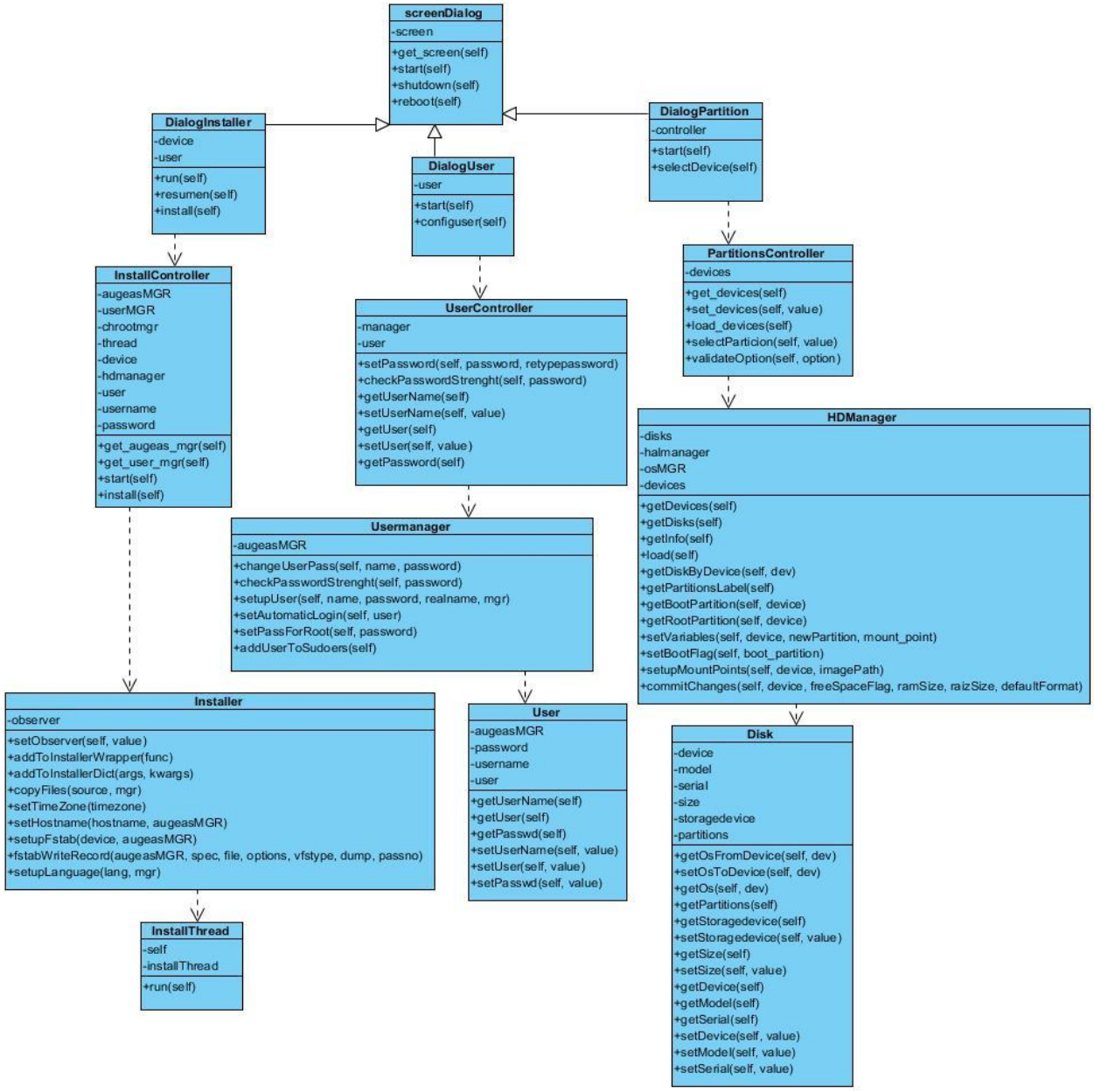



**Ilustración 10. Diagramas de Clases**

### 2.6.1 Diseño Arquitectónico.

En el desarrollo de la propuesta de solución se utiliza el patrón arquitectónico por capas. Es un modelo de desarrollo software, el objetivo primordial es la separación de las partes que componen un sistema software. De esta forma, por ejemplo, es sencillo y mantenible crear diferentes interfaces sobre un mismo sistema sin requerirse cambio alguno en la capa de datos o lógica. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que

sobrevenga algún cambio, solo afectará al nivel requerido sin tener que revisar entre el código fuente de otros módulos, dado que se habrá reducido el acoplamiento informático hasta una interfaz de paso de mensajes[24].

Para el desarrollo del instalador para GNU/Linux Nova de tipo kiosko se utilizó el patrón en tres capas, donde la primera se encarga de la presentación, la segundade la lógica del negocio y la tercera de los datos del sistema. La comunicación entre las capas está dada por el intercambio de información, pues todos los eventos de la capa de presentación implican acciones en la capa lógica, donde los datos de la misma lo contienen la capa de acceso de datos.

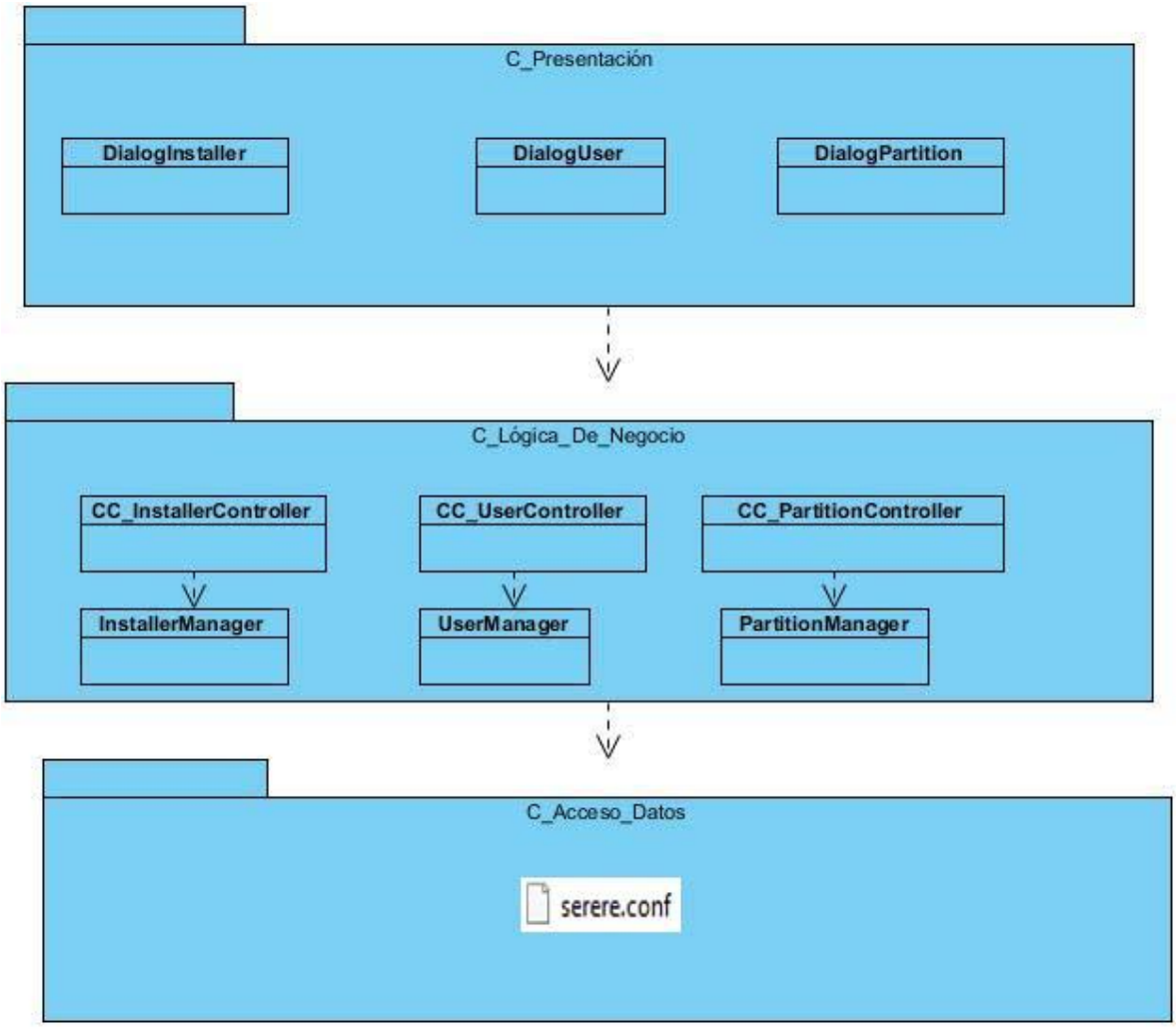



**Ilustración 11. Diagrama del diseño arquitectónico**

**Capa de presentación**

Esta capa es la que presenta el sistema al usuario, le comunica y captura la información en un mínimo de proceso. Esta capa se comunica únicamente con la lógica del negocio. También es conocida como interfaz de usuario y debe tener la característica de ser amigable para el usuario. En el mundo de la informática es conocida como interfaz gráfica. Su principal responsabilidad es mostrar información al usuario, interpretar los comandos de este y realizar algunas validaciones simples de los datos ingresados.

**Capa de lógica del negocio**

En esta capa es donde se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio o lógica del negocio porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de datos, y almacenar o recuperar datos de él.

**Capa de acceso a datos**

En esta capa es donde residen los datos y es la encargada de acceder a los mismos, recibe solicitudes de información desde la capa de negocio.

## 2.7. Conclusiones del capítulo

En este capítulo se dio una propuesta de solución que se ajustará a la situación problemática existente. Además, las historias de usuarios permitirán una mejor descripción de los requisitos con los que debe cumplir el sistema. Los requisitos del sistema y la arquitectura propuesta permitieron obtener el diseño de la aplicación. Se elaboró el diagrama de clases de la aplicación, lo que posibilitó mostrar cada una de las clases y las relaciones existentes entre ellas, con el objetivo de guiar la implementación de esta. También se seleccionaron los patrones de diseño y el modelo arquitectónico, los cuales optimizarán y harán más flexibles la implementación de la herramienta de instalación.

# CAPÍTULO 3: IMPLEMENTACIÓN, PRUEBAS Y EVALUACIÓN DEL INSTALADOR PARA A DISTRIBUCIÓN GNU/LINUX NOVA DE TIPO KIOSKO.

## 3.1. Introducción

En el presente capítulo se define el comportamiento de la herramienta de instalación, así como las restricciones del diseño, concebidas para la construcción de la aplicación y los estándares de codificación a seguir durante la implementación de sus funcionalidades, se describe el proceso de verificación y validación desarrollado para evaluar la propuesta de solución. Se verifica la calidad del resultado de la implementación, mediante los productos de trabajos generados durante las disciplinas de pruebas, exponiendo los resultados obtenidos.

## 3.2. Implementación

La implementación del sistema consiste en la traducción del diseño en una plataforma tecnológica, al definir plataformas se establecen los tipos de arquitectura, sistema operativo, lenguaje de programación o interfaz de usuario compatibles. El resultado es un producto de software con las características descritas en la actividad anterior, en esta actividad se obtiene un producto de software que puede ser utilizado por el cliente[25].

### 3.2.1. Modelado de la implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes podemos encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos[26].

**Diagrama de despliegue**

El diagrama de despliegue es utilizado para representar la distribución física de los componentes software en los distintos nodos físicos de la red. Con este diagrama, es posible modelar muchos de los aspectos hardware de un sistema con el nivel de detalle adecuado para que se pueda especificar la plataforma sobre la que se ejecutará, de tal forma que la frontera software - hardware del sistema sea mejor manejada[27].

El diagrama de despliegue de la herramienta de instalación para la distribución GNU/Linux de tipo kiosko presentado en la ilustración12 muestra el nodo (PC_Cliente necesario para la ejecución de la aplicación, en este caso computadoras), así como el componente instalador (herramienta de instalación a ejecutar).

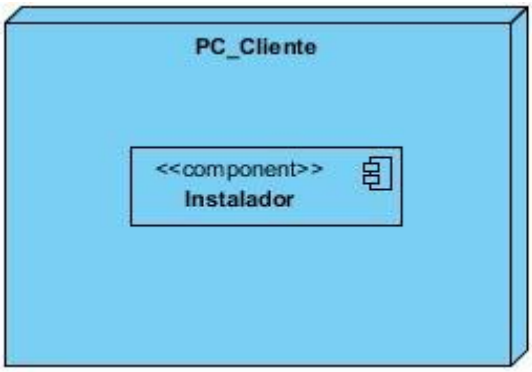



**Ilustración 12. Diagrama de despliegue**

## 3.3. Estándares de codificación

Las importaciones deben estar en líneas separadas, por ejemplo:

```
import time
import threading
import zmq
```

**Ilustración 13. Estándar de codificación 1**

Sin embargo, es correcto decir:

```
from dialog import Dialog
```

**Ilustración 14. Estándar de codificación 2**

- Evita usar espacios en blanco en las siguientes situaciones:
- Inmediatamente dentro de paréntesis, corchetes y llaves
- Inmediatamente antes de una coma, un punto y coma o dos puntos

```
def __init__(self):
    super(DialogInstaller, self).__init__()
    self.__device = None
    self.__user = None
```

**Ilustración 15. Estándar de codificación 3**

- Los nombres de las clases deben comenzar con mayúsculas. Para uso interno tienen un guión bajo como prefijo.

```
class DialogInstaller(screenDialog):
```

**Ilustración 16. Estándar de codificación 4**

- Las funciones deben ser en minúscula, (primera palabra en minúscula), aplicándose éstos tanto como sea necesario para mejorar la legibilidad.

```
def validateOption(self, option):
```

**Ilustración 17. Estándar de codificación 5**

Los estándares de codificación definidos en la investigación permiten tener un estilo único de codificación facilitando el estudio y entendimiento del código de la aplicación y garantizando que este sea más fácil de mantener.

## 3.4. Interfaz Gráfica de usuario del instalador para la distribución GNU/Linux Nova de tipo kiosko.

La interfaz de usuario es el medio de comunicación entre el usuario y el sistema y debe cumplir con las tres reglas siguientes: dar control al usuario, reducir la carga de memoria del usuario y debe ser consistente. El cumplimiento de estas reglas permite desarrollar interfaces flexibles, en las que el usuario no realice acciones innecesarias y no deseadas, así como la interrupción de una secuencia de acciones en el instante requerido por el usuario. Además, deben desglosar la información de forma progresiva a través de una estructura organizada, orientar al usuario en las tareas a desarrollar y que esas tareas se realicen en el contexto adecuado[28]. La ilustración18 muestra la interfaz de inicio y la interfaz de particionado para el proceso de instalación (ver en la ilustración19).

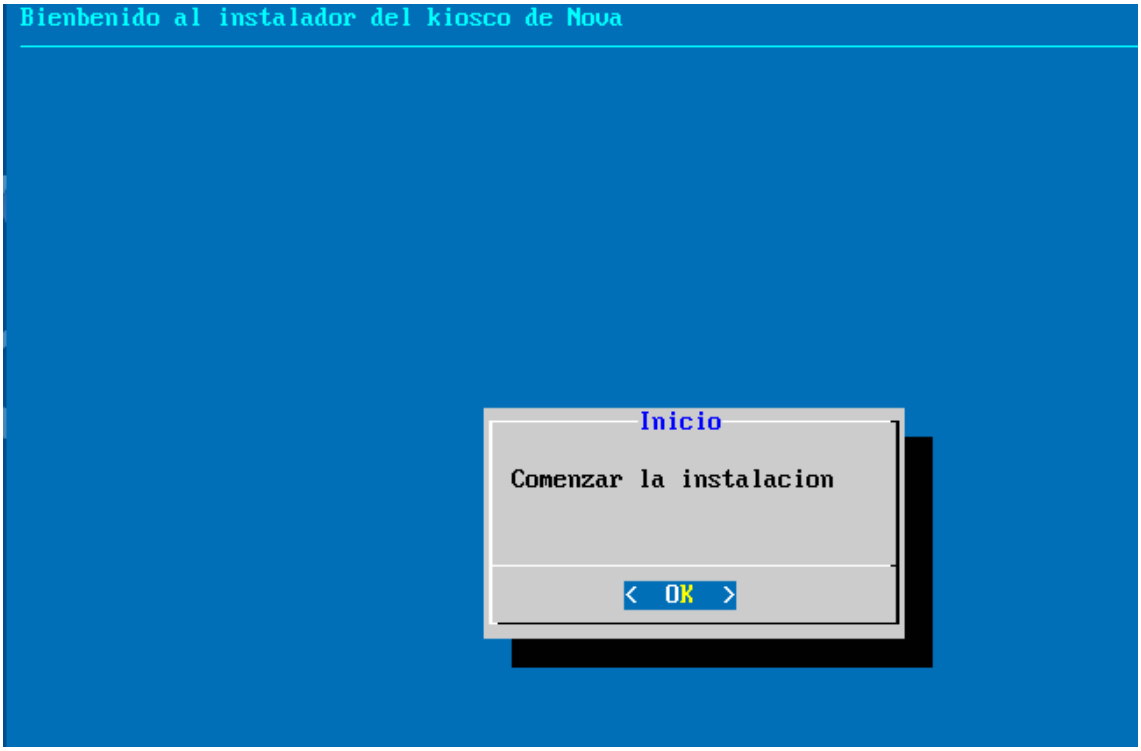



**Ilustración18. Interfaz de inicio**

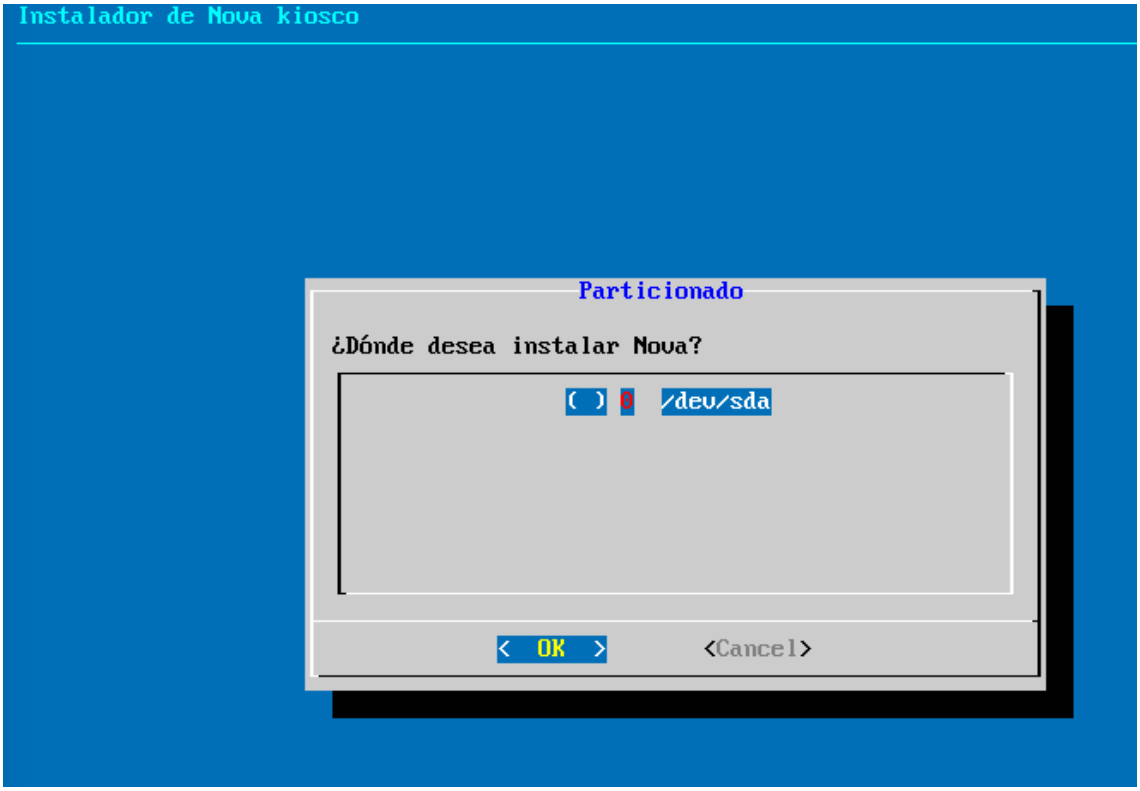



**Ilustración 19. Interfaz de Particionado**

## 3.5. Pruebas de Software

Las pruebas de software comprenden el conjunto de actividades que se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación, por medio de pruebas sobre el comportamiento del mismo[29].

### 3.5.1. Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas, desarrollando artefactos de prueba como diseños de casos de prueba[30]. A continuación, se presenta la prueba aplicada en esta disciplina.

**Pruebas unitarias**

Laspruebas unitariasson un método de pruebas de software que se realizan escribiendo fragmentos de código que testearáunidades de código fuente**.** El objetivo es asegurar que cada unidad funciona como debería de forma independiente[31].

El método de prueba utilizado para la realización de esta prueba es la Prueba de caja blanca. La técnica de prueba contenida en este método que se empleó fue la Técnica del camino básico.

## 3.6. Métodos y técnicas de pruebas

Métodos de pruebas utilizado:

Pruebas de caja blanca: verifican la correcta implementación de las unidades internas, las estructuras y sus relaciones y hacen énfasis en la reducción de errores internos[33]. Prueba de caja negra: verifican el correcto manejo de funciones externas provistas o soportadas por el software y que el comportamiento observado se apegue a las especificaciones del producto y a las expectativas del usuario[33].

Las técnicas de prueba utilizadas en la aplicación de los métodos de prueba fueron: Técnica de camino básico: es una técnica de prueba de caja blanca que permite obtener una medida de lacomplejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución independiente en un componente o programa[34]. Un camino o ruta es una vía por la cual procede la ejecución a través de una función desde su inicio hasta el fin[35].

Partición equivalente: permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

## 3.7. Aplicación y resultados de las pruebas realizadas al sistema

### 3.7.1. Pruebas internas

Las pruebas internas se desarrollaron a través de pruebas unitarias utilizando la técnica del camino básico de las pruebas de caja blanca. En esta técnica se utilizó la notación de grafo de flujo para representar en un grafo de flujo o del programa el código de la funcionalidad o procedimiento a probar. A continuación, se relacionan los pasos de la aplicación de la prueba del camino básico.

Pasos para la aplicación de la prueba de camino básico:

1. Dibujar el grafo de flujo de la funcionalidad o procedimiento a analizar.

```
def getDisks(self):
(1) context = pyudev.Context()
    devices = []

(2) for device in context.list_devices(subsystem='block'):
(3)     if self._is_disk(device):
            storage_device = 'storage_serial_' + str(device.get('ID_SERIAL'))
(4)         device_ = device.get('DEVNAME')
            model = device.get('ID_MODEL')  (6)      (7)
(5)         vendor = device.get('ID_VENDOR') or 'ATA'
            serial = device.get('ID_SERIAL')
            size = 0
(8)         removable = 0
            bus = device.get('ID_BUS')
            disk = Disk(device_, model, vendor, serial, size, storage_device, removable, bus)
            devices.append(disk)

(9) return devices
```

**Ilustración 20. Código de implementación de la funcionalidad Obtener Discos**

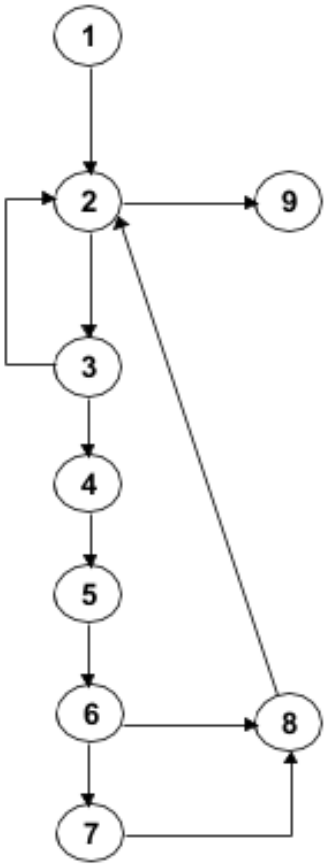



**Ilustración 21. Grafo de flujo del código de implementación de la funcionalidad Obtener Discos**

2. Determinar la complejidad ciclomática del grafo de flujo resultante.

   V(G) = A- N + 2

   V(G) = 11 – 9 + 2 = 4

   Dónde: A es el número de aristas del grafo de flujo

   N es el número de nodos del grafo

3. Determinar el conjunto básico de caminos linealmente independientes.

- Camino básico 1: 1- 2- 9
- Camino básico 2: 1-2-3-2-9
- Camino básico 3: 1-2-3-4-5-6-8-2-9
- Camino básico 4: 1-2-3-4-5-6-7-8-2-9

4. Definir los casos de prueba para comprobar la ejecución de cada camino del conjunto básico. En el diseño de los casos de prueba se debe especificar los siguientes elementos:

Descripción: Contiene la descripción sobre las restricciones de los datos de entradas que debe tener el caso de prueba.

- Condición de ejecución: se especifican los parámetros que debe poseer el caso de prueba para que se cumpla una condición deseada como respuesta del funcionamiento del procedimiento.
- Entrada: se muestran los parámetros de entrada al procedimiento.
- Resultados esperados: se explica el resultado esperado de la ejecución del procedimiento.

Las tablas 7 y 8 muestra los diseños de caso de pruebas para el camino 3 y 4 del conjunto básico de caminos literalmente independientes, correspondientes a la funcionalidad obtener discos.

**Tabla 7. Diseño de caso de prueba para el camino 3**

| **Diseño de caso de pruebas para el camino 3** | |
|---|---|
| **Descripción** | El sistema muestra los discos duros con el valor(ID_Vendor) establecido. |
| **Condición de ejecución** | El valor del disco duro ha sido establecido anteriormente. |
| **Entrada** | ID_Vendor = 1 |
| **Resultados esperados** | El sistema muestra los discos duros correctamente. |
| **Evaluación de la prueba** | Satisfactorio.<br><br>En este camino se prueba el mostrar discos duros de forma satisfactoria con el valor establecido. |

**Tabla 8. Diseño de caso de pruebas para el camino 4**

| **Diseño de caso de pruebas para el camino 4** |
|---|

| | |
|---|---|
| **Descripción** | El sistema muestra los discos duros que no tienen un valor establecido, donde los discos duros toman el valor de ATA por defecto. |
| **Condición de ejecución** | El valor del disco duro no ha sido establecido anteriormente. |
| **Entrada** | ID_Vendor = ATA |
| **Resultados esperados** | El sistema muestra los discos duros correctamente. |
| **Evaluación de la prueba** | Satisfactorio.<br>En este camino se prueba el mostrar discos duros sin un valor establecido tomando por defecto el valor de ATA de forma satisfactoria. |

A partir de la aplicación de los casos de prueba expuesto anteriormente se comprobó que el flujo de trabajo de las funcionalidades es correcto, ya que se comprobó que cada sentencia es ejecutada al menos una vez, cumpliéndose así las condiciones de la prueba y el resultado esperado es satisfactorio.

### 3.7.2. Pruebas funcionales

Las pruebas funcionales se desarrollaron utilizando la técnica partición de equivalencia del método de caja negra.

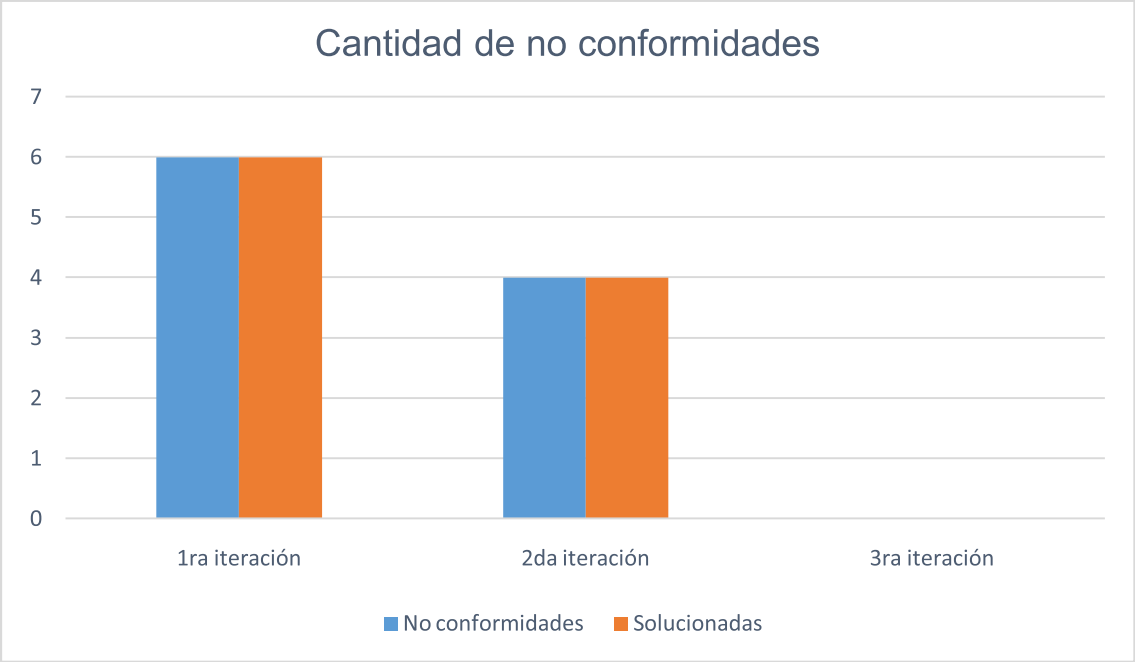



**Ilustración 22. Cantidad de no conformidades**

Las pruebas funcionales fueron realizadas en tres iteraciones en la primera se encontraron 5 no conformidades, 3 de funcionalidades y 2 de ortografía, en la segunda iteración se encontraron 2 no conformidades de funcionalidad, en la tercera iteración no se encontraron no conformidad. En la figura 21 se presenta un grafo con los resultados de las pruebas funcionales.

## 3.8. Conclusiones del capítulo

En este capítulo se aplicó la técnica Camino básico para comprobar que el flujo de trabajo de las funcionalidades es correcto, ya que cada sentencia es ejecutada al menos una vez y la técnica de Partición equivalente verificó en tres iteraciones realizadas que el sistema cumple con los requisitos definidos.

# CONCLUSIONES GENERALES

La investigación realizada cumple con los objetivos planteados mediante el desarrollo del instalador para la distribución GNU/Linux Nova de tipo kiosko y se arriba a las siguientes conclusiones:

El análisis de los referentes teóricos y de los sistemas informáticos estudiados evidenció la necesidad de desarrollar el instalador para la distribución GNU/Linux Nova de tipo kiosko para agilizar, dinamizar y guiar al usuario en el proceso de instalación y la configuración del sistema.

Se obtuvo el instalador para la distribución GNU/Linux Nova de tipo kiosko que permite guiar al usuario en el proceso de instalación y la configuración del sistema.

La validación de la investigación se realizó a partir de la aplicación de técnicas, métricas y pruebas que garantizaron el correcto funcionamiento del instalador y demostraron la satisfacción del cliente hacia el instalador desarrollado.

# RECOMENDACIONES

1. Que el producto se generalizado en toda la Universidad de las Ciencias Informáticas.

2. Brindar soporte a la aplicación con el objetivo de mantener su correcto funcionamiento.

# BIBLIOGRAFÍA REFERENCIADA

[1] C. laudia Y. Paz, «Informatización de la sociedad en Cuba», jul. 2019.

[2] E. Castillo Noda, «Nova, la distribucción GNU/Linux hecha en Cuba».

[3] N. Guardia, «Porteus Kiosk», oct. 2015.

[4] J. Pacheco, «Agilizar el proceso de instalación», sep. 2017.

[5] J. Barrios, «Qué es un instalador», feb. 2017.

[6] M. E. Raffino, «Concepto de Sistema Operativo», ago. 2017.

[7] Á. Robleado, «Python: Características, evolución y futuro», sep. 2019.

[8] R. Menas Martínez, «Interfaz de Líneas de Comandos», 2015.

[9] «Que son las interfaz o GUI», sep. 2017.

[10] J. A. Castillo, «Partición GTP», ene. 2019.

[11] Y. Martínez, «Qué es UEFI y porqué es importante para la seguridad de tu equipo», 2018.

[12] M. Rivera Soltero, «Formato MS DOS», abr. 2017.

[13] «Portal: YaST», dic. 2015.

[14] E. Martínez, «Ubiquiti-Instalar UNMS sobre Ubuntu Server», ago. 2018.

[15] «Instalar con el instalador de Debian», jul. 2019.

[16] T. Rodríguez Sanchéz, «Metodología de desarrollo para la actividad productiva de la UCI», 2015.

[17] F. Ruiz, «Lenguaje Unificado de Modelado -UML», 2015.

[18] «Visual Paradigm», feb. 2016.

[19] R. Shecter, «Pythondialog», dic. 2018.

[20] San Agustín Del Valle, «Planeación y etapas», 2015.

[21] Delgado Dapena, «MODELO CONCEPTUAL DE UN PROYECTO DE SOFTWARE», 2015.

[22] F. San, «Especificación de requisitos», jun. 2018.

[23] Janneth Amaya, «Qué es un patrón de diseño», sep. 2016.

[24] Peláez Juan, «Arquitectura basada en capas», may 2015.

[25] Leandro Alegsa, «Implementación de sistemas», jun. 2016.

[26] Jorge Guardarrama, «Modelado de Implementación», jun. 2015.

[27] Johana Sarmiento, «Diagrama de despliegue», abr. 2015.

[28] Juan Andrés Corralez, «Interfaz de usuario», ago. 2018.

[29] Juan Quijano, «Pruebas de software», mar. 2018.

[30] Jorge Hernán Abad Londoño, «Tipos de pruebas de software», abr. 2015.

[31] Oscar Moreno, «Pruebas unitarias», ago. 2019.

[32] Mera Paz, «Pruebas de aceptación del software», ago. 2016.

[33] CristelRamirez Carbona, «Métodos de software caja blanca y ccaja negra», mar. 2015.

[34] Cesar Javier PerezPincay, «Ingeniería de software, método caja blanca y técnica camino básico», may 2015.

[35] Cesar Javier PerezPincay, «Prueba de camino básico», jun. 2015.

# ANEXOS

## Anexo 1. Cuestionario

| **Encuesta** | |
|---|---|
| ¿Considera factible la nueva versión del instalador para la distribución GNU/Linux de tipo kiosko? | _No _Sí _No Se |
| ¿Considera negativo algún elemento de la herramienta de instalación para la distribución GNU/Linux Nova desarrollada? | _No _Sí _No Se |
| ¿La forma en que se configura el sistema con la herramienta de instalación para la distribución GNU/Linux Nova de tipo kiosko permite satisfacer todas las necesidades existentes? | _No _Sí _No Se |
| Luego de haber visto la herramienta de instalación para distribución GNU/Linux Nova de tipo kiosko refleje en qué medic gusta la solución desarrollada. | _Me gusta mucho<br>_Me gusta más de lo que me disgusta<br>_Me da lo mismo<br>_Me disguta más de lo que me gusta<br>_No me gusta nada<br>_No sé |

**Anexo 2. Historias de usuario**

**Tabla 10.HU Configurar Usuario**

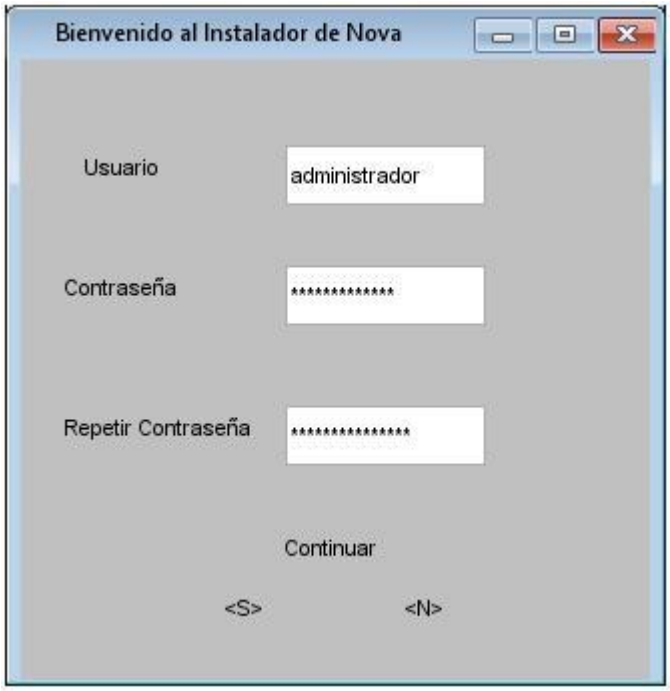
| | |
|---|---|
| **Número:** HU1 | **Requisito**: Configurar usuario |
| **Programador:** Miraida González Ponce | **Iteración Asignada:** 1 |
| **Prioridad**: Media | **Tiempo Estimado:** 48horas |
| **Riesgo en Desarrollo**: Ausencia del desarrollador por enfermedad o pérdida de información imprescindible . | **Tiempo Real:** 52 horas |
| **Descripción:** El usuario inserta su identificador como administrador (Ejemplo: administrador), contraseña en el formulario de autenticación para acceder a las funcionalidades del sistema y debe repetir contraseña. | |
| **Observaciones:** Si el sistema no identifica al usuario le mostrará una notificación sobre este problema | |
| **Prototipo elemental de interfaz gráfica de usuario:**   | |

**Tabla 1. HU Mostrar discos duros.**

| | |
|---|---|
| **Número:** HU2 | **Requisito**: Mostrar discos duros |
| **Programador: Miraida** González Ponce | **Iteración Asignada:** 1 |
| **Prioridad**: Media | **Tiempo Estimado:** 72 horas |
| **Riesgo en Desarrollo**: Ausencia del desarrollador por enfermedad o pérdida de información imprescindible . | **Tiempo Real:** 52 horas |
| **Descripción:** El usuario selecciona el disco duro en que se realizará la instalación | |
| **Observaciones:** Si el usuario no selecciona el disco duro para el proceso de instalación le mostrará una notificación sobre este problema | |

**Tabla 2. HU Eliminar tabla de particiones**

| | |
|---|---|
| **Número:** HU3 | **Requisito**: Eliminar tabla de particiones |
| **Programador:** Miraida González Ponce | **Iteración Asignada: 1** |
| **Prioridad**: Alta | **Tiempo Estimado:** 72 horas |
| **Riesgo en Desarrollo**: Ausencia del desarrollador por enfermedad o pérdida de información imprescindible . | **Tiempo Real:** 52 horas |
| **Descripción:** El usuario elimina la tabla de particiones que se obtiene en el proceso de instalación | |
| **Observaciones:** Si el usuario no elimina la tabla de partición correcta el sistema le mostrará una notificación sobre este problema | |

**Tabla 3. HU Crear la tabla de particiones.**

| | |
|---|---|
| **Número:** HU4 | **Requisito**: Crear la tabla de particiones |
| **Programador:** Miraida González Ponce | **Iteración Asignada:** 1 |
| **Prioridad**: Media | **Tiempo Estimado:** 72 horas |

| | |
|---|---|
| **Riesgo en Desarrollo**: Ausencia del desarrollador por enfermedad o pérdida de información imprescindible . | **Tiempo Real:** 52 horas |
| **Descripción:** El usuario crea la nueva tabla de particiones | |
| **Observaciones:** Si el usuario no crea la tabla de particiones correctamente el sistema le mostrará una notificación sobre este problema | |

**Anexo 3. Interfaces de usuario.**


**Ilustración 23. Resumen de la instalación**

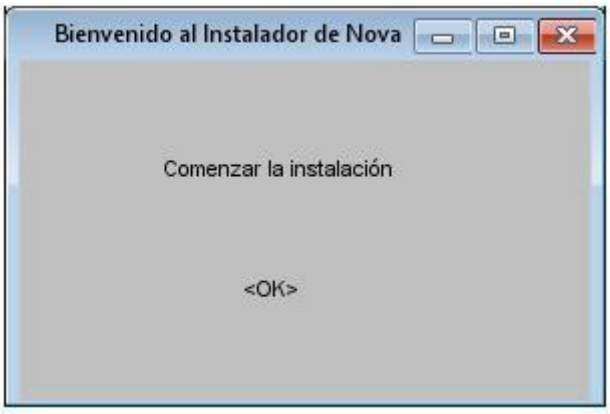



**Ilustración 24. Bienvenido al instalador**

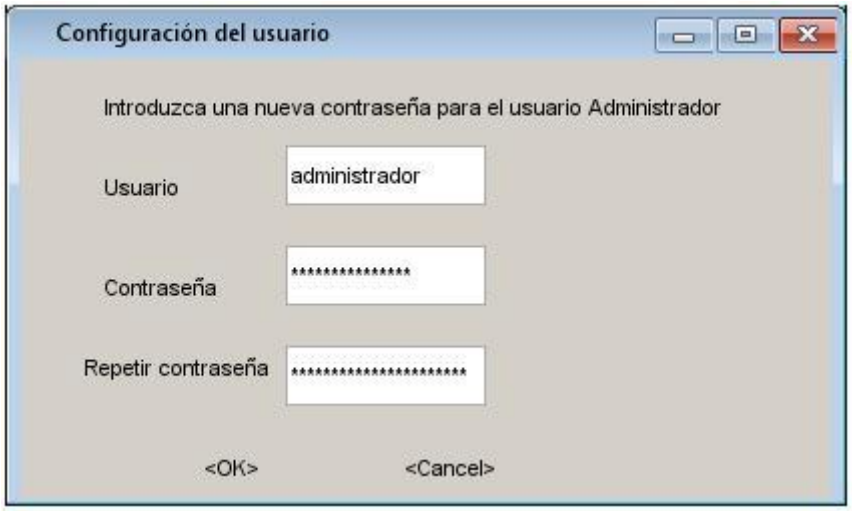



**Ilustración 25. Bienvenido al instalador**