# Universidad de las Ciencias Informáticas


*"Módulo para el monitoreo en tiempo real de clientes ligeros desde Nova-LTSP "*

**Trabajo de diploma para optar por el título de**

**Ingeniero en Ciencias Informáticas**

**Autor:**

Lexys Manuel Díaz Alonso

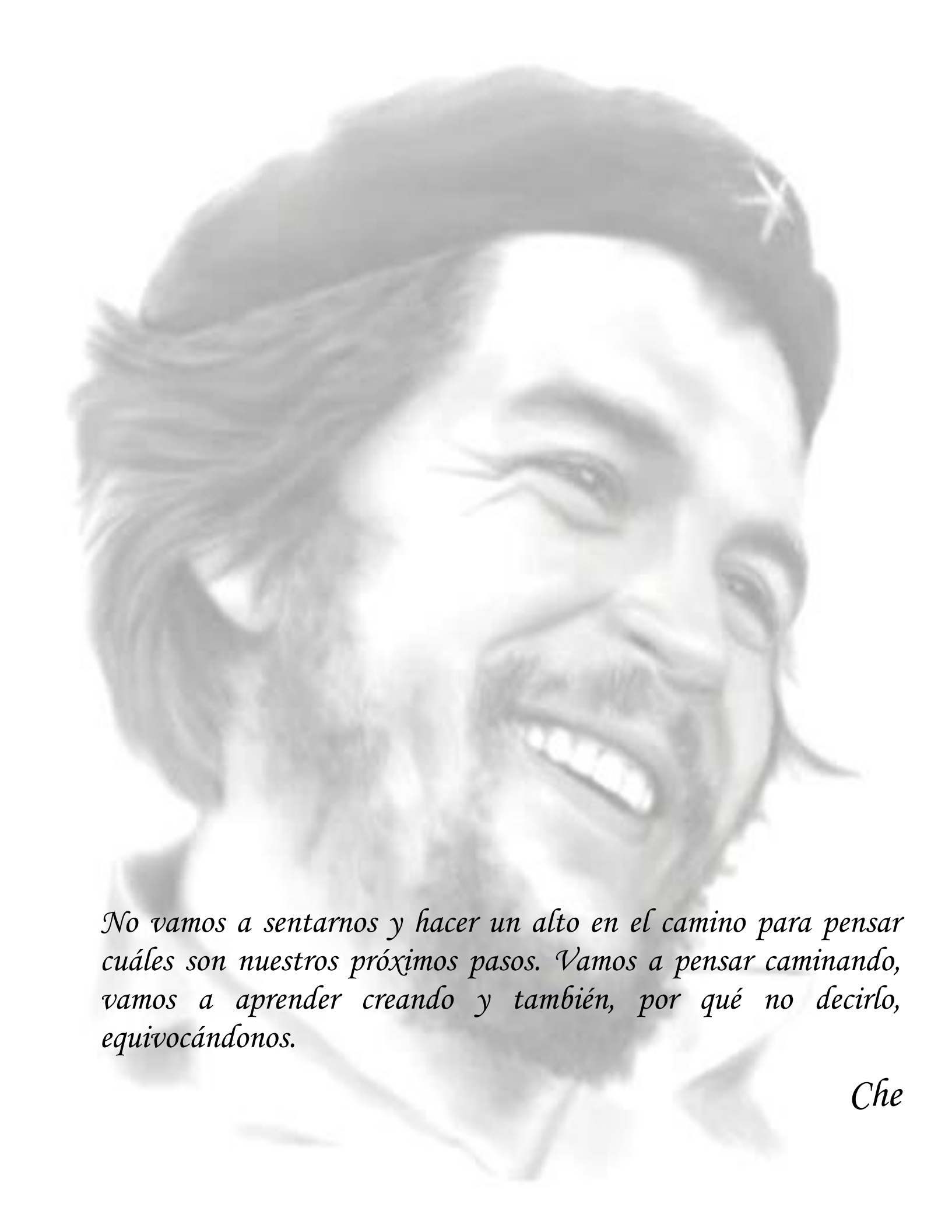
**Tutores:**

Ing. Yasiel Pérez Villazón

Ing. Lisdania de la Caridad Delgado Olivera

La Habana, junio de 2017

Año 59 de la Revolución



*No vamos a sentarnos y hacer un alto en el camino para pensar cuáles son nuestros próximos pasos. Vamos a pensar caminando, vamos a aprender creando y también, por qué no decirlo, equivocándonos.*

*Che*

## *Declaración de Autoría*

Declaro por este medio que yo Lexys Manuel Díaz Alonso, con carné de identidad 93102611328 soy el autor principal del trabajo titulado “Módulo para el monitoreo en tiempo real de clientes ligeros desde Nova-LTSP ” 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 firmo la presente a los ___ días del mes de junio del año 2017.

_____

Firma del Autor

Lexys Manuel Díaz Alonso

_____ _____

Firma de la Tutora

Ing. Yasiel Pérez Villazón

Firma de la Tutora

Ing. Lisdania de la Caridad Delgado Olivera

# *Dedicatoria*

*A mi mamá Mirian, a quién debo todo lo que soy, por su amor, dedicación, confianza y enseñarme a construir mi propio futuro.*

*A Dios, por ser mi refugio, mi fortaleza, por resaltar en mi sentimientos de gratitud y amor.*

*A mi mamá, por ser la persona más especial de mi vida, por su esmero, comprensión y apoyarme en todas mis decisiones, te quiero mucho.*

*A mi tío Israel por acogerme como su hijo, por su cariño y apoyo constante.*

*A mi papá, por su preocupación y siempre confiar en mí.*

*A mis tíos Conrado y Carmen, por la ayuda que me han brindado.*

*A Lisdy por todo su amor, dedicación, siendo siempre incondicional y muy especial en mi vida.*

*A mi familia, por amarme y guiarme por el camino correcto.*

*A mis amigos, en especial a Silvio , Raúl , Edel , Roberto, Sobrino, Elvis, Javier, Liset y Yojahny, por los buenos momentos que hemos compartido juntos.*

*A Yuli y a Victor, por su inmensa ayuda en el momento más difícil de la universidad, mil gracias y bendiciones para sus vidas.*

*A José Angel y Alejandro, por todas sus enseñanzas y estar siempre disponible durante estos 5 años de la carrera.*

*A mis compañeros de aula, por todo lo que aprendí de ustedes y las cosas buenas que me llevo de cada uno.*

*A mis tutores Yasiel y Lisdania, por confiar en mí y por su ayuda incondicional en el desarrollo de esta investigación.*

*A los profesores de la facultad, por su tiempo y haber contribuido en mi formación como profesional.*

*A todos los que me brindaron su ayuda.....gracias!*

## Resumen

La Distribución Cubana GNU/Linux Nova, cuenta con la plataforma de administración de clientes ligeros Nova-LTSP, que brinda un conjunto de funcionalidades para administrar estos clientes. Sin embargo tiene limitantes para el monitoreo en tiempo real de los clientes ligeros, tarea que resulta engorrosa para los administradores de red, debido a que se realiza a través de líneas de comandos. Además no permite tener un control de los escritorios, lo que implica que los usuarios en ocasiones realicen un uso innecesario de aplicaciones. Debido a esta situación, la presente investigación tuvo como objetivo: Desarrollar un módulo que permita el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, para favorecer el control de los recursos de *hardware* en el servidor. La propuesta de solución estuvo guiada por la metodología de desarrollo de *software* Variación AUP-UCI, se seleccionó para la implementación Python 2.7 como lenguaje de programación sobre el *framework* Django en su versión 1.10. Se utilizó como sistema gestor de base de datos PostgreSQL 9.4 y como entorno integrado de desarrollo PyCharm2016 en su versión 1.4. Esta solución permite a los administradores conocer el consumo de la RAM, la SWAP y la CPU, además de finalizar los procesos que ejecuta el usuario, para favorecer el control de estos recursos en el servidor. Las pruebas realizadas arrojaron como resultado que el módulo implementado responde a los requerimientos definidos por el cliente. El método de criterio de expertos demostró que la propuesta de solución, favorece el control de los recursos de *hardware* en el servidor.

**Palabras clave**: clientes ligeros, control, módulo, monitoreo, Nova-LTSP

## Índice de Contenido

## Índice de Figuras

## Índice de Tablas

## Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) se han desarrollado aceleradamente, lo que ha provocado un elevado auge en el desarrollo de la sociedad. Su evolución ha generado un cambio sustancial en la forma de producir, gestionar y acceder a la información.

La informatización de la sociedad se define en Cuba como el proceso de utilización ordenada y masiva de las TIC para satisfacer las necesidades de información y conocimiento de todas las personas y esferas de la sociedad (1). Dentro de los principios fundamentales de las bases y prioridades para el perfeccionamiento de la informatización de la sociedad en Cuba se encuentra la informatización en función de desarrollar y modernizar coherentemente todas las esferas de la sociedad y apoyo a las prioridades del país; la sostenibilidad y soberanía tecnológica (2).

Sin embargo, como resultado de la política de bloqueo económico y financiero que el gobierno de Estados Unidos mantiene sobre Cuba, y ante el dominio de Microsoft sobre el mercado internacional de sistemas operativos, para Cuba resulta insostenible mantenerse a la par del modelo consumista impuesto por las economías capitalistas en el área de las TIC. Existe además el riesgo que implica para la seguridad y la soberanía nacional la dependencia absoluta de productos extranjeros que día a día juegan un papel cada vez más importante en las esferas gubernamentales, productivas y sociales del país. De ahí que sea un requisito necesario lograr la soberanía tecnológica y apostar por el uso del *software* libre y el código abierto.

Cuba, en medio de dificultades y contradicciones que afectan el proceso de informatización , se dedica a la búsqueda de nuevas alternativas. En el año 2004 por convenio del Consejo de Ministros se anuncia el acuerdo 084, que orienta una migración paulatina de los Órganos y Organismos de la Administración Central del Estado (OACE) hacia aplicaciones de *software* libre (3). La migración hacia estándares de *software* libre y código abierto es una de las tareas fundamentales que se realiza con el objetivo de eliminar las dependencias tecnológicas. En tal sentido, la Universidad de las Ciencias Informáticas (UCI) juega un papel fundamental.

La UCI tiene la misión de producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación y servir de soporte a la industria cubana de la informática (4). Dentro de su infraestructura docente-productiva se encuentra el Centro de *Software* Libre (CESOL) que tiene

como objetivos fundamentales desarrollar la Distribución Cubana GNU/Linux Nova y definir las directrices, lineamientos y soluciones que guiarán la migración nacional (5).

Esta distribución cubana, desde su creación, sirve de apoyo para el proceso de migración del país, por lo que cuenta con un equipo de especialistas multidisciplinario y con una buena preparación que se encarga de su constante desarrollo y perfeccionamiento. También permite la gestión y el despliegue de clientes ligeros. Estos son computadoras clientes o un *software* de cliente en una arquitectura de red Cliente-Servidor que depende del servidor central para las tareas de procesamiento, y se enfoca en transportar la entrada y la salida entre el usuario y el servidor remoto (6).

El cliente ligero sólo se ocupa de las funciones básicas como la pantalla, teclado, el ratón y el sonido. El *hardware* utilizado por ellos puede ser pequeño y barato. El costo de mantenimiento de los clientes ligeros es muy bajo, duran más tiempo dado que no tienen almacenamiento con partes móviles como pueden ser los discos duros. Si se avería un cliente ligero no se pierde la información, porque ésta está almacenada en el servidor. Simplemente se cambia el cliente por otro y se continúa trabajando. Si el cliente ligero es robado la información no terminará en manos de un tercero (6). En el servidor de terminales se ejecutan todas las aplicaciones y contiene todos los datos. Todo el mantenimiento habitual tales como actualizaciones de *software*, administración, se lleva a cabo en el servidor de terminales (6).

La Distribución Cubana GNU/Linux Nova, cuenta con la plataforma de administración de clientes ligeros Nova-LTSP, que brinda un conjunto de funcionalidades para administrar estos clientes como son diagnóstico del sistema, detalles de la imagen de sistemas operativos y asignaciones de red; sin embargo tiene limitantes para conocer los recursos de *hardware* que están consumiendo los clientes ligeros en el servidor, tarea que resulta engorrosa para los administradores de red, debido a que se realiza a través de líneas de comandos y los mismos, en muchos casos, no poseen la experiencia necesaria para realizarlos. Dicha herramienta no permite tener un control de los escritorios, lo que implica que los usuarios en ocasiones realicen un uso innecesario de aplicaciones, en consecuencia se aumenta el consumo de los recursos en el servidor y se afecta su funcionamiento. Además el proceso de verificar el estado ya sea Encendido/Apagado requiere de mucho tiempo, puesto que los administradores realizan este proceso a cada uno de los clientes ligeros, trabajo que se vuelve lento si se tienen muchos.

En correspondencia con lo antes expuesto se plantea como **problema de investigación:** ¿Cómo contribuir al monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, para que se favorezca el control de los recursos de *hardware* en el servidor?.

Se identifica como **objeto de estudio** de la investigación: El proceso de administración de los clientes ligeros.

Para darle solución al problema anteriormente planteado, se define como **objetivo general:** Desarrollar un módulo que permita el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, para que se favorezca el control de los recursos de *hardware* en el servidor.

Se define como **campo de acción:** El proceso de monitoreo en tiempo real de los clientes ligeros manejados por Nova-LTSP.

Para cumplir la meta propuesta se han trazado los siguientes **objetivos específicos:**

1. Construir el marco teórico conceptual y estudiar el estado del arte respecto a las herramientas de administración de los clientes ligeros.
2. Diseñar el módulo de monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP.
3. Implementar el módulo de monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP.
4. Validar el correcto funcionamiento del módulo implementado.

Luego de haber realizado una revisión bibliográfica y desarrollado el marco teórico, se formula la siguiente **hipótesis de investigación:** el módulo que permite el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, favorece el control de los recursos de *hardware* en el servidor. Teniendo en cuenta la hipótesis anteriormente planteada, se define como **variable independiente:** el módulo que permite el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, el cual consiste en una aplicación informática que brinda un conjunto de funcionalidades para el monitoreo en tiempo real de los clientes ligeros. Como **variable dependiente:** el control de los recursos de *hardware* en el servidor. Esta variable hace alusión al consumo de los recursos de *hardware* de los clientes ligeros en el servidor.

Para dar cumplimento a los objetivos específicos se definieron las siguientes **tareas de investigación.**

1. Estudio de los conceptos asociados al marco teórico de la investigación.

2. Caracterización de las herramientas de administración de los clientes ligeros.
3. Definición de la arquitectura de la propuesta de solución.
4. Elaboración de los artefactos requeridos por la metodología de desarrollo seleccionada.
5. Desarrollo de las funcionalidades que permitan el monitoreo en tiempo real de los clientes ligeros.
6. Validación de la propuesta de solución.
7. Validación de la hipótesis de la investigación.

Para facilitar el cumplimiento del objetivo propuesto y de las tareas de investigación se emplean **métodos teóricos y empíricos** de la investigación científica.

**Métodos teóricos:**

**Histórico-Lógico:** Es utilizado en el análisis de los sistemas homólogos, de manera que permita buscar elementos que los caractericen y aspectos para fundamentar la propuesta de solución a la problemática planteada.

**Analítico-Sintético:** Con el fin de descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.

**Modelación:** Utilizado en la representación, mediante el uso de diagramas, de las características del sistema, y relaciones entre objetos que intervienen en los procesos implementados por la propuesta de solución.

**Métodos empíricos**

**Entrevista:** Se realizó una entrevista al cliente para obtener información acerca de las funcionalidades que debía cumplir el módulo.

El presente documento de trabajo de diploma está compuesto por introducción, tres capítulos, conclusiones generales, recomendaciones, referencias bibliográficas, anexos y glosario de términos.

**Capítulo 1: Soporte teórico para la implementación de un módulo de monitoreo de clientes ligeros:** Se definen los principales conceptos asociados al dominio del problema que son indispensables para el desarrollo y comprensión de esta investigación. Además se definen las herramientas y tecnologías que se van a utilizar en el desarrollo de la aplicación.

**Capítulo 2: Características y diseño del módulo:** Se describe la propuesta de solución para resolver el problema planteado y se especifican los requisitos funcionales y no funcionales. Por otra parte, se describen las historias de usuario correspondientes a los requisitos funcionales, la arquitectura y los patrones de diseño que se utilizarán durante la implementación.

**Capítulo 3: Implementación y pruebas al módulo:** Se documentan los artefactos asociados a la implementación de la propuesta de solución y se realizan los casos de prueba para lograr el desarrollo de un *software* con la calidad requerida. Además, se establecen los estándares de codificación que se tuvieron en cuenta para el desarrollo del sistema.

## Capítulo1. Soporte teórico para la implementación de un módulo de monitoreo de clientes ligeros

En el presente capítulo se abordan los conceptos fundamentales que sustentan la presente investigación. Se realiza un estudio de las herramientas de administración de los clientes ligeros, para finalmente seleccionar los aspectos a tener en cuenta en la propuesta de solución, según el resultado obtenido en la entrevista aplicada al cliente. Además, se definen las herramientas, lenguajes de programación, tecnologías y metodología de desarrollo a utilizar.

Para lograr un mejor entendimiento del problema a investigar se tuvieron en cuenta conceptos importantes tales como: control y monitoreo, clientes ligeros y arquitectura cliente servidor. Estos elementos guiarán el desarrollo del módulo a implementar. A continuación se definen los mismos:

### 1.1 Control y monitoreo

En la actualidad, los sistemas de control y monitoreo a distancia se han convertido en factor común de diversos campos de aplicación que van desde la medicina hasta la seguridad incluyendo las redes informáticas. Con el amplio crecimiento de las redes informáticas surgió la necesidad de controlar todo el tráfico de la red (7).

Entre las principales características en el control y monitoreo de redes se pueden encontrar las siguientes:

- ✓ Posibilitar la revisión del tráfico de la red en tiempo real.
- ✓ Administrar y controlar equipos y aplicaciones de forma remota.
- ✓ Analizar el rendimiento.
- ✓ Gestionar los problemas que se presenten tanto en las redes locales como en las remotas.
- ✓ Conservar y almacenar datos de la red para manejar reportes.

### 1.2 Clientes ligeros

Un cliente ligero es un ordenador cliente, conectado a una red cliente-servidor que depende principalmente del servidor central para las tareas de procesamiento. Este servidor se encarga de distribuir los escritorios virtuales entre todos los clientes. Basa su eficiencia en la utilización de los recursos mínimos para su funcionamiento, un cliente ligero no procesa ningún tipo de dato, por lo que no se requiere una máquina potente, dejando ese trabajo al equipo servidor (8).

A continuación, se muestran las ventajas y desventajas de los clientes ligeros (9) .

**Ventajas**

**Menores costos de administrativos de IT[1]:** Los clientes ligeros son manejados casi enteramente en el servidor. El *hardware* tiene menos lugares donde puede fallar, el entorno local es altamente restringido, y el cliente es más simple y a menudo carece de almacenamiento permanente, proporcionando protección contra el *malware*[2].

**Seguridad de datos mejorada:** Si un dispositivo del cliente ligero sufre un serio desperfecto de trabajo, no se perderá ningún dato, puesto que residen en el servidor de terminales y no en el dispositivo de punto de operación.

**Más bajos costos de *hardware*:** El *hardware* del cliente ligero es generalmente más barato porque no contiene disco duro, memoria de aplicaciones, o un procesador potente. Generalmente también tienen un período más largo antes de requerir una mejora o llegar a ser obsoletos. Hay menos piezas móviles y se actualiza o mejora el servidor y la red en lugar de los clientes, porque la limitación en su desempeño es la resolución de pantalla que tiene un ciclo de vida muy largo.

**Menos consumo de energía:** El *hardware* dedicado de cliente ligero tiene mucho más bajo consumo de energía que los típicos PC de clientes pesados, ahorran hasta un 80% de electricidad y cuidan el medio ambiente. Esto no sólo reduce los costos de energía en los sistemas de computación, en algunos casos puede significar que los sistemas de aire acondicionado no son requeridos o no necesitan ser actualizados lo que puede ser un ahorro de costos significativo y contribuir a alcanzar los objetivos en ahorro de energía. Sin embargo, son requeridos servidores más potentes y sistemas de comunicaciones.

**Desventajas**

[1] IT: Tecnología Informática

[2] Malware: Término que engloba a todo tipo de programa o código informático malicioso cuya función es dañar un sistema o causar un mal funcionamiento.

**Más requerimientos del servidor:** Un servidor de cliente pesado no requiere tan alto nivel de desempeño como un servidor de cliente ligero. Esto resulta en servidores más caros.

**Inapropiado para conexiones de red pobres:** Los clientes ligeros pueden ser inusualmente lentos, sobre una conexión de red de alta latencia. Por otra parte, no trabajan en absoluto cuando la red está caída. Con un cliente pesado, puede ser posible trabajar fuera de línea, aunque la manera orientada a red en la que mucha gente trabaja hoy en día, significa que el uso del cliente pesado también puede ser restringido si la red está caída.

**Más difícil de reutilizar:** Mientras que un cliente pesado puede ser usado en aplicaciones de cliente ligero, cuando el *hardware* se vuelve obsoleto para el uso como cliente pesado, debido a que es estándar y puede operar de una manera autónoma, el *hardware* de cliente ligero es más difícil de revender o reutilizar para otro propósito en caso de que sea retirado.

## 1.3 Arquitectura cliente-servidor

Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es un solo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivos y los servidores de correo. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma (10).

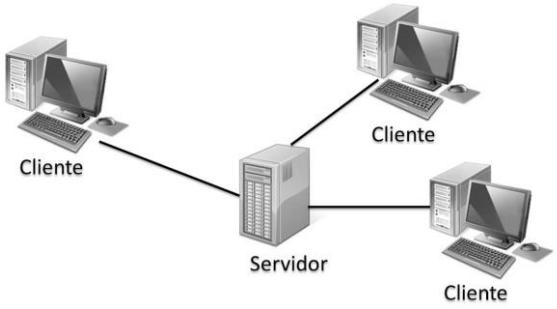



Figura 1: Arquitectura cliente servidor (10)

Los clientes ligeros se pueden implementar en dos modalidades diferentes: con procesamiento del lado del servidor o con procesamiento del lado del cliente. Cada una de estas modalidades ofrece ventajas y desventajas que la hacen apropiada para un entorno específico.

**Procesamiento del lado del servidor**

**Ventajas**

- ✓ Los ordenadores de clientes puede ser de bajas prestaciones ya que el procesamiento se hace en el servidor.
- ✓ La administración de los clientes es menos complicada puesto que todos trabajan directamente en el servidor.
- ✓ Posibilita una tasa alta de procesamiento en dependencia de las características del *hardware* del servidor y la cantidad de clientes conectados.
- ✓ El tráfico por la red es menor en comparación con la modalidad del procesamiento del lado del cliente lo que lo hace factible en redes de poco ancho de banda.

**Desventajas**

- ✓ Soporta un límite de conexiones bastante bajo directamente proporcional a las capacidades de *hardware* del servidor puesto que cada cliente tiene que conectarse remotamente al servidor para poder funcionar.
- ✓ Si una aplicación congela[3] el sistema del servidor se ven afectados todos los clientes.

[3] Congela: Hace referencia a cuando el sistema por algún motivo se paraliza impidiendo su funcionamiento.

- ✓ Demanda mayor responsabilidad del administrador con respecto a configuraciones internas del servidor u otros datos sensibles como pueden ser las versiones del *software* instalado.

**Procesamiento del lado del cliente**

**Ventajas**

- ✓ Es capaz de soportar gran número de conexiones en dependencia de la capacidad de la red.
- ✓ No se requiere que el servidor posea un *hardware* muy potente, cualquiera máquina promedio puede funcionar como un servidor.
- ✓ Permite aprovechar el *hardware* de los clientes para el procesamiento de datos.

**Desventajas**

- ✓ Hace que la carga de la red sea mayor puesto que por cada cliente es necesario establecer dos conexiones con el servidor.

El uso de una modalidad u otra está en dependencia de los objetivos con los que se quieran usar y con el *hardware* tanto del lado del servidor como del lado del cliente que se tenga. Por ejemplo si se cuentan con clientes ligeros con buenas prestaciones y servidores de bajas prestaciones es más adecuado utilizar la modalidad del procesamiento de lado del cliente. En otro caso donde las computadoras clientes no sean muy potentes y los servidores consten con un *hardware* potente es más factible aplicar la modalidad del procesamiento del lado del servidor.

Después de haber analizado los conceptos fundamentales para entender los principales temas de la investigación; como parte del marco teórico de la investigación se hace necesario realizar un análisis de las herramientas de administración de los clientes ligeros.

## 1.4 Herramientas para la administración de los clientes ligeros

Existen distintas herramientas para la administración de clientes ligeros. La presente investigación se enmarca en las herramientas libres y se exponen las características para hacer una valoración crítica.

**TCos *(Thin Client Operating System)***

TCos es un conjunto de herramientas tanto para el arranque de terminales ligeros como para su control, distribuido bajo la licencia GNU/GPL[4] de *software* libre. Con TCos se puede arrancar equipos sin disco, desde la red, basado en el kernel, binarios y librerías del sistema servidor. Dentro de sus principales herramientas podemos encontrar TcosMonitor.

TcosMonitor es un módulo usado para monitorizar y administrar los terminales de la red a partir del servidor o de un terminal.

La función de TCos es integrar los terminales con el servidor, gestionar los procesos ejecutados por cada usuario, generar la imagen del sistema operativo para ser cargada en los terminales, además de permitir la reutilización de los ordenadores obsoletos para ser usados como terminales (11).

**Características:**

- ✓ Monitorización y administración local de los terminales a través del TcosMonitor.
- ✓ Monitorización y administración remota de los terminales a través de TcosPHPMonitor[5].
- ✓ Suporte a múltiples terminales conectados simultáneamente al mismo servidor.
- ✓ Control remoto del volumen del sonido emitido en el terminal.
- ✓ Permite la reutilización de ordenadores obsoletos para ser usados como terminales, reduciendo los costes de la red, prolongando su vida útil y reduciendo así considerablemente el impacto ambiental de los equipos.

[4] GNU/GPL: Es una licencia que da la libertad de usar, estudiar, compartir (copiar) y modificar el *software*.

[5] TcosPHPMonitor: Es una alternativa a TCosMonitor permitiendo administrar los terminales de la red, a través de un sistema Web, desarrollado en PHP.

***Lan Core***

Es una herramienta para la gestión de los clientes ligeros. Proporciona información actualizada de los terminales conectados, acceso a su configuración y control sobre la ejecución de los terminales. Además se encuentra bajo la licencia GNU/GPL (12).

**Características:**

- ✓ Permite la configuración de los terminales conectados:
  - Dirección del servidor y protocolo utilizado.
  - Nombre de usuario y clave de acceso.
  - Resolución del monitor: refresco horizontal y vertical.
  - Idioma del teclado
- ✓ Tiene un control absoluto sobre sus terminales
  - Puede comprobar cuando están conectadas.
  - Apagar o reiniciar remotamente los terminales.

***OpenThinClient***

Es una solución de código abierto y libre junto con un componente administrativo basado en una interfaz gráfica. Está pensado para ambientes donde se debe montar y administrar una cantidad media o grande de clientes ligeros de manera eficiente. Se encuentra bajo la licencia GNU/GPL (13).

**Características:**

- ✓ Tiene una filosofía de centralizar los servicios en un servidor determinado lo cual minimiza el trabajo administrativo del lado del servidor, donde se controlarán las terminales.
- ✓ El inicio y la configuración de los clientes ligeros son implementados usando tecnologías estándares como LDAP[6],DHCP[7], TFTP[8] y NFS[9].

[6] LDAP (Protocolo Ligero de Acceso a Directorios): Es un protocolo a nivel de aplicación (según el modelo OSI) que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.

- ✓ Los clientes ligeros solo necesitan una interfaz PXE[10] para iniciar la imagen. Todos los datos de configuración se guardan en una base de datos LDAP que viene junto con el servidor *Open Thin Client.*

***LTSP (Linux Terminal Server Project)***

Son un conjunto de aplicaciones servidores que proporcionan la capacidad de ejecutar GNU/Linux en computadoras de pocas prestaciones, ya sea de velocidad o de bajo costo. Se distribuye bajo la licencia GNU/GPL de *software* libre.

LTSP posee una herramienta administrativa llamada *Thin Client Manager* la cual permite realizar varias acciones tales como: ejecutar un programa en el cliente, desconectar un cliente, enviar un mensaje, bloquear y desbloquear la pantalla de un cliente, ver y agrupar a los usuarios y editar las restricciones de los usuario.

Su funcionamiento consiste en repartir a través de la red un kernel de GNU/Linux que es ejecutado por los clientes que posteriormente ejecutarán secuencias de *scripts*[11] típicos de una pequeña distribución. Los clientes podrán acceder a las aplicaciones por medio de una consola textual o por un servidor gráfico que se comparte utilizando el protocolo XDMCP[12] (14).

---

[7] DHCP (Protocolo de Configuración Dinámica de Host): Es un protocolo de red que permite a los clientes de un a red IP obtener sus parámetros de configuración automáticamente.

[8] TFTP (Protocolo de transferencia de archivos trivial): es un protocolo de transferencia muy simple semejante a una versión básica de FTP.

[9] NFS (Sistema de archivos de red): Es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local.

[10] PXE (Entorno de ejecución de prearranque): Es un entorno para arrancar e instalar el sistema operativo en computadoras a través de una red, de manera independiente de los dispositivos de almacenamiento de datos disponibles (como discos duros) o de los sistemas operativos instalados.

[11] Scripts: En informática es un guión o conjunto de instrucciones que permiten la automatización de tareas creando pequeñas utilidades

[12] XDMCP (Protocolo de Control de Administrador de la Pantalla X): es un protocolo para compartir archivos en una red. Funciona como una conexión de escritorio remoto a la máquina servidora

**Características**

- ✓ Fácil administración: Con una infraestructura centralizada, la administración de los archivos, las aplicaciones y demás características deben ser configuradas en el servidor, evitando efectuar tareas repetitivas de configuración sobre los clientes.
- ✓ Escalabilidad: Para instalar nuevos terminales clientes simplemente se debe configurar en el servidor una cuenta de usuario, conectar el cliente ligero a la red y encenderlo, y se tendrá acceso a todas las aplicaciones disponibles en el servidor.
- ✓ Información protegida en el servidor.

**Resultados del estudio de las herramientas de administración de los clientes ligeros**

A raíz del estudio realizado, se ha demostrado que las herramientas no satisfacen los requerimientos establecidos por el cliente. Las herramientas LTSP y *LanCore* no permiten el monitoreo en tiempo real del consumo de los recursos de *hardware* de los clientes ligeros en el servidor. *OpenThinClient* carece de un control remoto de los escritorios de cada terminal y TCos no es la utilizada por la plataforma de administración de los clientes ligeros Nova-LTSP a la cual se debe integrar la propuesta de solución. Sin embargo, aunque ninguno de estos sistemas constituye una solución integral al problema en cuestión, su estudio permitió obtener una visión más amplia de la solución y proporcionó funcionalidades de interés para enriquecer la solución propuesta, como por ejemplo tener un control absoluto sobre sus terminales, comprobar cuando están conectadas, ver y finalizar los procesos ejecutados por cada usuario y apagar remotamente los clientes ligeros. Lo mencionado anteriormente enfatizó la necesidad de desarrollar un módulo para monitorear en tiempo real los clientes ligeros para el proyecto, que responda a las necesidades planteadas.

A continuación se explican las principales características del protocolo SSH[13] y el porqué de su uso, con el propósito de garantizar la seguridad de los datos en el módulo a desarrollar.

[13] Intérprete de Órdenes Seguro.

## 1.5 Seguridad de los datos. Protocolo SSH (Secure SHell)

SSH es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación remota tales como FTP[14] o Telnet[15], SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas (15).

El protocolo SSH proporciona los siguientes tipos de protección:

- ✓ Después de la conexión inicial, el cliente puede verificar que se está conectando al mismo servidor al que se conectó anteriormente.
- ✓ El cliente puede transmitir su información de autenticación al servidor, como el nombre de usuario y la contraseña, en formato cifrado.
- ✓ Todos los datos enviados y recibidos durante la conexión se transfieren por medio de encriptación fuerte, lo cual los hacen extremamente difícil de descifrar y leer.
- ✓ El cliente tiene la posibilidad de reenviar aplicaciones X11[16]desde el servidor. Esta técnica, llamada reenvío por X11, proporciona un medio seguro para usar aplicaciones gráficas sobre una red.
- ✓ Si el servidor usa la técnica del reenvío de puerto, los protocolos considerados como inseguros (POP[17], IMAP[18]), se pueden cifrar para garantizar una comunicación segura

**¿Por qué usar SSH?**

[14] FTP (Protocolo de Transferencia de Archivos) :Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP(Protocolo de Control de Transmisión), basado en la arquitectura cliente-servidor.

[15] Telnet: Es el nombre de un protocolo de red que nos permite acceder a otra máquina para manejarla remotamente.

[16] X11: Se refiere al sistema de visión por ventanas X11R6.7, tradicionalmente llamado Sistema de ventanas X o simplemente X. Red Hat Enterprise Linuxcontiene XFree86, un sistema de ventanas X de código abierto.

[17] POP (Protocolo de oficina de correos): Es un protocolo estándar de internet de la capa aplicación en el Modelo OSI.

[18] IMAP (Protocolo de acceso a mensajes de Internet): Es un protocolo de aplicación que permite el acceso a mensajes almacenados en un servidor de Internet.

Los usuarios tienen a su disposición una variedad de herramientas que les permiten interceptar y redirigir el tráfico de la red para ganar acceso al sistema. En términos generales, estas amenazas se puede catalogar del siguiente modo:

- ✓ Intercepción de la comunicación entre dos sistemas: En este escenario, existe un tercero en algún lugar de la red entre entidades en comunicación que hace una copia de la información que pasa entre ellas. La parte interceptora puede interceptar y conservar la información, o puede modificar la información y luego enviarla al recipiente al cual estaba destinada.
- ✓ Personificación de un determinado *host*: Con esta estrategia, un sistema interceptor finge ser el recipiente a quien está destinado un mensaje. Si funciona la estrategia, el sistema del usuario no se da cuenta del engaño y continúa la comunicación con el *host* incorrecto.

Ambas técnicas interceptan información potencialmente confidencial y si esta intercepción se realiza con propósitos hostiles, el resultado puede ser adverso.

Si se utiliza SSH para inicios de sesión de *shell* remota y para copiar archivos, se pueden disminuir estas amenazas a la seguridad notablemente. Esto es porque el cliente SSH y el servidor usan firmas digitales para verificar su identidad. Adicionalmente, toda la comunicación entre los sistemas cliente y servidor es encriptada. No servirán de nada los intentos de falsificar la identidad de cualquiera de los dos lados de la comunicación ya que cada paquete está cifrado por medio de una llave conocida sólo por el sistema local y el remoto.

## 1.6 Metodología de desarrollo de *software*

Las metodologías de desarrollo de *software* son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de *software*. Van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando qué personas deben participar en el desarrollo de las actividades y qué papel deben tener. Además detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla (16).

Visto esto se puede concluir que, en el desarrollo de *software* es importante la selección de una adecuada metodología para lograr el éxito del producto final. A continuación, se describe la metodología seleccionada para guiar el proceso de desarrollo de *software* de la presente investigación:

**AUP-UCI** es una variación de la metodología AUP[19], de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. En aras de aumentar la calidad del *software* que se produce esta metodología se apoya en el Modelo CMMI-DEV v1.3. (por sus siglas en inglés *Capability Maturity Model Integration Development*, Integración de Modelos de Madurez de Capacidades para Desarrollo). El mismo constituye una guía para aplicar las mejores prácticas y obtener un producto o servicio con calidad en una entidad desarrolladora (17).

**Fases Variación AUP-UCI**

**Inicio:** El objetivo de esta fase es llevar a cabo las actividades relacionadas con la planeación del proyecto. Se realiza un estudio inicial de la organización que actúa como cliente y se obtiene información clave acerca del alcance del proyecto, se realizan estimaciones de tiempo y esfuerzo, y finalmente se decide si se ejecuta o no.

**Ejecución:** En esta etapa se ejecutan las actividades requeridas para desarrollar el *software*. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elabora la arquitectura y el diseño, se implementa y se libera el producto.

**Cierre:** En esta fase se analizan tanto los resultados del proyecto como su ejecución y se llevan a cabo las actividades formales de cierre de proyecto.

Se selecciona la metodología AUP-UCI pues se adapta al ciclo de vida definido para la actividad productiva de la UCI, y da la posibilidad al cliente de siempre acompañar al equipo de desarrollo para convenir los requisitos y así poder implementarlos. Además, fue aprobada cuando la institución se sometió al proceso de certificación CMMI nivel 2 para el desarrollo de *software*.

### 1.7 Herramientas y tecnologías

Las herramientas son aplicaciones, programas o meramente instrucciones que ofrecen la posibilidad de realizar determinadas funcionalidades con disímiles propósitos. Mientras que, las tecnologías son el conjunto de conocimientos técnicos, ordenados científicamente que permiten diseñar y crear bienes y servicios (18). A continuación, se muestran todas las herramientas y tecnologías utilizadas en el desarrollo

[19] Proceso Unificado Ágil, del inglés *Agile Unified Process*.

de la solución propuesta, estas fueron seleccionadas por políticas del proyecto al que pertenece el sistema que se pretende desarrollar. Se fundamentan también otras características de importancia.

### 1.7.1 Lenguaje de modelado

"El modelado constituye una simplificación de la realidad donde se define lo esencial para la construcción del software con los objetivos de comunicar la estructura de un sistema complejo, especificar el comportamiento deseado del sistema, comprender mejor lo que se está desarrollando y descubrir oportunidades de simplificación y reutilización" (19).

**Lenguaje Unificado de Modelado (UML)**

El Lenguaje Unificado de Modelado es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de *software*. Captura las decisiones y los conocimientos sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre tales sistemas (20).

Para llevar a cabo el diseño del módulo se propone utilizar UML, en su versión 2.0, por las siguientes características (20):

- ✓ **Visualizar:** Permite expresar de una forma gráfica un sistema de forma tal que otro lo pueda entender.
- ✓ **Especificar:** Permite especificar cuáles son las características de un sistema antes de su construcción.
- ✓ **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- ✓ **Documentar:** Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Las ventajas del lenguaje UML son (21):

- ✓ Es estándar, lo que significa que facilita la comunicación.
- ✓ Está basado en el meta modelo con una semántica bien definida.
- ✓ Se basa en una notación gráfica concisa y fácil de aprender y utilizar.

- ✓ Se puede utilizar para modelar *software* en distintos dominios: sistemas de información empresariales, sistemas web, sistemas críticos y de tiempo real.
- ✓ Es fácilmente extensible.

### 1.7.2 Herramienta CASE

Dentro de las herramientas claves para el desarrollo de aplicaciones informáticas se encuentran las herramientas de Ingeniería de *Software* Asistida por Computadora (por sus siglas en inglés *Computer-Aided Software Engineering*, CASE). Este tipo de herramienta se utiliza para ayudar a actividades del proceso de *software* tales como la ingeniería de requisitos, el diseño, el desarrollo de aplicaciones y las pruebas (21).

**Visual Paradigm**

Visual Paradigm es una herramienta UML que permite modelar el ciclo de vida completo del desarrollo de un *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite construir todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (22).

Se empleará Visual Paradigm, en su versión 8.0, para modelar los diagramas que se generen en cada una de las etapas del proceso de desarrollo de *software*, además de brindar las siguientes ventajas (23).

- ✓ **Dibujo:** Facilita el modelado de UML, proporciona herramientas específicas para ello. Esto también permite la estandarización de la documentación, ya que se ajusta al estándar soportado por la herramienta.
- ✓ **Corrección sintáctica**: Controla que el modelado sea correcto
- ✓ **Coherencia entre diagramas:** Dispone de un repositorio común, es posible visualizar el mismo elemento en varios diagramas, esto evita las duplicidades.
- ✓ **Integración con otras aplicaciones:** Permite integrarse con otras aplicaciones como herramientas ofimáticas, lo cual aumenta la productividad.
- ✓ **Trabajo multiusuario:** Permite el trabajo en grupo, proporciona herramientas de compartición de trabajo.

- ✓ **Reutilización:** Dispone de una herramienta centralizada donde se encuentran los modelos utilizados para otros proyectos.
- ✓ **Generación de código:** Permite generar código de forma automática, esto reduce los tiempos de desarrollo y evita errores en la codificación del *software*.
- ✓ **Generación de informes:** Permite generar diversos informes a partir de la información introducida en la herramienta.

### 1.7.3 Lenguaje de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar procesos que pueden ser ejecutados por máquinas computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones (24).

**Python**

Python es un lenguaje de programación fuerte y fácil de aprender, además posee un enfoque de programación orientada a objetos. Este lenguaje cuenta con estructuras de datos eficientes y de alto nivel. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para el desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas (25).

Se selecciona el lenguaje de programación Python, en su versión 2.7, debido a que es el utilizado por la plataforma de administración de clientes ligeros Nova-LTSP a la que se debe integrar la presente propuesta de solución, por lo que su utilización contribuye a la homogeneidad y compatibilidad del código, además de poseer las siguientes ventajas (26):

- ✓ **Portabilidad:** Funciona en casi todos los sistemas operativos.
- ✓ **Fácil:** Es muy fácil comenzar a desarrollar programas en Python. Su sintaxis clara, gracias a la notación identada (con márgenes), hace al código cómodo de leer.
- ✓ **Poder:** Posee extensiones escritas en el propio lenguaje de Python que permiten, entre otras cosas, el acceso a la base de datos, la edición de audio y video, interfaz gráfica de usuario y el desarrollo web.

- ✓ **Código abierto:** Python es un lenguaje de código abierto, lo que significa que se puede utilizar libremente.

### 1.7.4 Servidor web

Un servidor web es un programa que se ejecuta continuamente en un computador, manteniéndose a la espera de peticiones de ejecución realizadas por un cliente. El servidor web se encarga de contestar a estas peticiones de forma adecuada a través del protocolo HTTP (por sus siglas en inglés *Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto), entregando como resultado una página web. El servidor web también contiene intérpretes de diferentes lenguajes de programación, estos son los encargados de ejecutar el código embebido dentro del HTML de las páginas web antes de enviar el resultado al cliente (27).

**Apache**

Apache es uno de los servidores web más utilizados en la red. Es uno de los mayores triunfos del *software* libre, pues logró vencer a competidores como *Netscape y Microsoft*, como plataforma de servidores web. Es un servidor modular, multiplataforma, extensible, popular (fácil de conseguir ayuda) y gratuito (27).

Para el desarrollo del módulo se decide usar el servidor web Apache, en su versión 2.2, debido a las características que evidencia:

- ✓ Provee una muy buena base para la seguridad del sistema, gracias a los módulos de Autenticación, Autorización y Control de Acceso al Servidor Web.
- ✓ La extensibilidad por módulos lo hace extremadamente flexible y fácil de usar, así como de configurar.

### 1.7.5 Entorno de desarrollo integrado

Un entorno de desarrollo integrado, o bien conocido como IDE (por sus siglas en inglés *Integrated Development Environment*), “es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios” (28).

**PyCharm IDE**

PyCharm IDE es un IDE para el lenguaje de programación Python que posee un juego completo de herramientas para el desarrollo productivo. Para llevar a cabo la implementación del módulo se hará uso de PyCharm2016, en su versión 1.4.

PyCharm ofrece las siguientes funciones (28):

- ✓ Auto-completamiento de código.
- ✓ Señalamiento de errores con soluciones fáciles.
- ✓ Posibilita una fácil navegación para proyectos y código.
- ✓ Mantiene el código bajo control de chequeos, asistencia de pruebas, refactorizaciones y un conjunto de inspecciones que posibilitan codificar de forma limpia y sostenible.

### 1.7.6 Sistema Gestor de Bases de datos

Un Sistema gestor de base de datos (SGBD) es el *software* que permite la utilización y/o actualización de los datos almacenados en una o varias bases de datos, por uno o varios usuarios desde diferentes puntos de vista. El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manipular los datos, de forma que no le sea necesario conocer el modo de almacenamiento ni el método de acceso empleado (29).

**PostgreSQL**

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[20] y con su código fuente disponible libremente. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (30).

Se selecciona el SGBD PostgreSQL9.4, debido a que fue el SGBD utilizado en la plataforma de administración de clientes ligeros Nova-LTSP a la cual se va a integrar el presente módulo, por lo que su utilización contribuye a la homogeneidad de la información, además de las facilidades antes mencionadas.

[20] BSD: Distribución de Software de Berkeley. Este tipo de licencia permite a los programadores utilizar, modificar y distribuir a terceros el código fuente del software original.

### 1.7.7 Marcos de trabajo

Un marco de trabajo o *framework* no solo es utilizado en el ámbito de aplicaciones web, se pueden encontrar *frameworks* para el desarrollo de aplicaciones médicas, de visión de computador y para el desarrollo de juegos. Por lo que al referirse a este término se hace alusión a una estructura de *software* compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación (31).

**Django**

Django es un *framework* que impulsa el desarrollo de código limpio al promover buenas prácticas de desarrollo web, sigue el principio DRY (conocido también como Una vez y sólo una). Usa una modificación de la arquitectura Modelo-Vista-Controlador (MVC), llamada MTV (Model – Template – View), que sería Modelo-Plantilla-Vista (32).

Django ofrece las siguientes facilidades (33):

- ✓ Sistema de plantillas para separar la presentación de un documento de sus datos.
- ✓ Construcción automática de interfaces de administración.
- ✓ Vistas genéricas que recogen ciertos estilos y patrones comunes en su desarrollo y los abstraen, de modo que se puede escribir rápidamente vistas comunes de datos sin tener que escribir mucho código.
- ✓ Sistema para que no tengan que ser recalculadas cada vez que se piden.
- ✓ Integración con bases de datos y aplicaciones existentes.
- ✓ Construcción de aplicaciones multilenguaje permitiendo especificar cadenas de traducción de más de cuarenta idiomas.

Para el desarrollo del módulo se decidió usar Django, en su versión 1.10, pues es un marco de trabajo de desarrollo web totalmente implementado sobre Python con el que se pueden crear y mantener aplicaciones de alta calidad.

**Bootstrap**

Bootstrap es una colección de herramientas de *software* libre para la creación de sitios y aplicaciones web. Contiene plantillas de diseño basadas en HTML y CSS con tipografías, formularios, botones,

gráficos, barras de navegación y demás componentes de interfaz, así como extensiones opcionales de JavaScript. Fue desarrollado por Mark Otto y Jacbod Thornton de Twitter, como un *framework* para fomentar la consistencia a través de herramientas internas. Es de código abierto, además de ser compatible con la mayoría de los navegadores web (34).

Para el desarrollo del módulo se utiliza Bootstrap, en su versión 3.0, porque se integra además con la biblioteca JQuery y está basado en herramientas actuales y potentes como CSS3 y HTML5.

**JQuery**

JQuery es un *framework* JavaScript que permite de una manera ágil crear aplicaciones web, es libre, su código se distribuye bajo la Licencia del Instituto Tecnológico de Massachusetts (por sus siglas en inglés MIT) y la Licencia Pública General de GNU en su versión 3. Cuenta con funcionalidades para acceder al árbol del DOM [21] del documento HTML, posee selectores para elegir los elementos por clases e identificadores, o para seleccionar por tipo de elemento. Permite el manejo de los eventos que generan los periféricos de entrada como el mouse o ratón y el teclado. Es multiplataforma y extensible a través de plugins [22], permitiendo hacer más amplias sus funcionalidades, o agregar aquellas que no estén en existencia. Su tamaño es realmente pequeño, disminuyendo el ancho de banda o tráfico de red, desde el servidor al dispositivo cliente. Posee una amplia documentación en línea[23] (35).

Para el desarrollo del módulo se decidió utilizar JQuery, en su versión 1.10.3, para las validaciones no funcionales en la parte del cliente, por ser un marco de trabajo que facilita la selección de elementos HTML, la creación de animaciones y evita la implementación de funcionalidades comunes.

Además se empleó la estrategia marcaria de los productos que serán desarrollados en los diferentes centros de la Universidad de las Ciencias Informáticas, específicamente la línea de desarrollo Xilema que permite elaborar interfaces gráficas con JQuery y Bootstrap.

[21] DOM: Modelo de objeto del documento.

[22] Plugins: Componente de software que permite ampliar las funcionalidades del mismo.

[23] http://docs.jquery.com.

## 1.8 Conclusiones parciales

En este capítulo se han abordado los elementos teóricos que dan sustento a la propuesta de solución del problema planteado, arribando a las siguientes conclusiones:

- ✓ Las relaciones existentes entre los principales conceptos asociados al dominio de la presente investigación permitieron una mayor comprensión de la propuesta de solución.
- ✓ El análisis de las diferentes herramientas de administración de los clientes ligeros, permitió determinar las características que constituyen la base para el diseño de las funcionalidades que se definen en la propuesta de solución.
- ✓ La selección de la metodología, herramientas y tecnologías posibilitó obtener la base tecnológica de la propuesta de solución. Para guiar el proceso de desarrollo del módulo se adoptó como metodología de desarrollo de *software* AUP-UCI. Se definió utilizar la herramienta CASE Visual Paradigm 8.0 para el modelado de los artefactos del análisis y diseño de la solución. Por otra parte, para la implementación se utilizó Python 2.7 como lenguaje de programación sobre el *framework* Django en su versión 1.10, apoyado en el IDE PyCharm2016 en su versión 1.4.

## Capítulo 2. Características y diseño del módulo

En el presente capítulo se define la manera en que se construye el sistema, o sea, se plasman sus características a través de los requisitos funcionales y no funcionales, las historias de usuarios que especifican cada requisito funcional, se describe la propuesta de solución donde se incluye la arquitectura del *software* para conocer la manera en que se estructura, los patrones de diseño a utilizar, el diagrama de clases, el diagrama de paquetes y por último el diagrama de despliegue.

A continuación se expone la propuesta del módulo para el monitoreo de clientes ligeros, donde se describen las principales funcionalidades que el mismo va a poseer.

### 2.1 Propuesta del módulo para el control y monitoreo de clientes ligeros.

En la presente investigación se propone el desarrollo de un módulo que permita, desde Nova-LTSP, monitorear los clientes ligeros en tiempo real. La aplicación garantizará la seguridad de los datos, mediante conexiones seguras empleando el protocolo SSH.

Dentro de la sección Clientes de la plataforma Nova-LTSP se encontrarán todas las funcionalidades correspondientes al monitoreo de los clientes ligeros. El módulo permitirá adicionar un nuevo cliente ligero insertando determinadas características tales como: Nombre, Dirección MAC, Dirección IP y Descripción. Permitirá además, editar un cliente ligero cambiando las características que desee, así como eliminar un cliente. En el sistema se mostrará un listado de los clientes ligeros que han sido adicionados con el objetivo de que el administrador pueda monitorizarlos. También ofrecerá el estado Encendido/Apagado de los clientes ligeros para saber cuántos clientes están consumiendo recursos en el servidor. Para ello el módulo contará con la opción “Detalles del cliente ligero” donde se muestra el consumo de la RAM, la SWAP, y la CPU, además de los procesos que ejecuta el usuario, con la opción de poder finalizar los procesos seleccionados previamente. Al mismo tiempo posibilitará tener una vista del escritorio a través de la conexión remota y de esta forma conocer las aplicaciones que utiliza el usuario.

### 2.2 Modelo de dominio

Un modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. También se le denomina modelo conceptual, modelo de objetos del dominio y modelo de objetos de análisis. Al utilizar la notación UML, un modelo de dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación. Su principal objetivo es

definir las interrelaciones de los objetos más importantes representados mediante clases. Además, desempeña un papel clave en la comprensión del entorno actual (36).

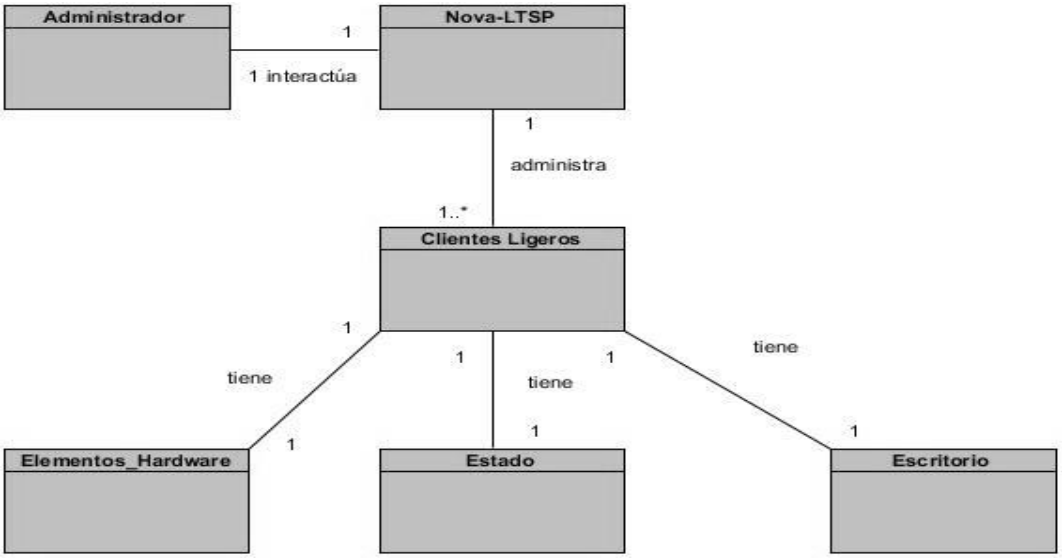



Figura 2: Modelo de dominio.

**Descripción de los conceptos del dominio**

**Administrador:** entidad que representa a la persona que interactúa y posee control total sobre la plataforma Nova-LTSP.

**Nova-LTSP:** entidad que representa a la plataforma de administración de los clientes ligeros.

**Clientes_Ligeros:** entidad que representa a los clientes ligeros.

**Elementos_*Hardware*:** entidad que representa los elementos de *hardware* que consumen los clientes ligeros.

**Estado:** entidad que representa el estado de los clientes ligeros.

**Escritorio:** entidad que representa el escritorio de los clientes ligeros.

## 2.3 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 (17).

### 2.3.1 Fuentes para la obtención de requisitos

Las fuentes para la obtención de requisitos utilizadas fueron:

- ✓ Modelo de dominio
- ✓ Análisis de las herramientas existentes.
- ✓ Especialistas de CESOL.

### 2.3.2 Técnicas para la 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 de una aplicación, permiten investigar aspectos generales para posteriormente ser especificados con un mayor detalle, requieren ser adecuadamente orientadas para cubrir la información que se requiere capturar (21).

**Entrevista**

La entrevista es de gran utilidad para obtener información cualitativa como opiniones, o descripciones subjetivas de actividades. Es una técnica muy utilizada, y requiere una mayor preparación y experiencia por parte del analista. La entrevista consiste en un “intento sistemático de recoger información de otra persona” a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada donde es muy importante la forma en que se plantea la conversación y la relación que se establece en la entrevista. Se realizó una entrevista al cliente para obtener información acerca de las funcionalidades que debía cumplir el módulo.

## *2.4* Especificación de los requisitos de *software*

El propósito de la definición de requisitos es especificar las condiciones o capacidades con que el sistema debe contar y las restricciones bajo las cuales debe operar, logrando un acuerdo entre el equipo de

desarrollo y el cliente y especificando las necesidades reales de forma que satisfaga sus expectativas (37).

En la solución propuesta se identificaron 10 requisitos funcionales y 8 requisitos no funcionales. Los cuales se muestran a continuación:

### 2.4.1 Requisitos funcionales

Los requisitos funcionales definen el comportamiento interno de un *software*, son condiciones que el sistema ha de cumplir. Estos muestran las funcionalidades que deben satisfacerse para cumplir con las especificaciones de *software* (38).

La siguiente tabla muestra el listado de los requisitos funcionales identificados para el desarrollo del módulo y la descripción de los mismos.

Tabla 1: Listado de requisitos funcionales.

| No. | Nombre del requisito funcional | Descripción |
|---|---|---|
| **Prioridad** | | **Alta** |
| RF1 | Adicionar cliente ligero. | Adiciona un cliente ligero insertando determinadas características. |
| RF2 | Editar cliente ligero. | Edita un cliente ligero cambiando las características que se desee. |
| RF3 | Eliminar cliente ligero. | Elimina un cliente ligero. |
| RF4 | Conectar con el escritorio de los clientes ligeros. | Accede de forma remota a los clientes ligeros. |
| **Prioridad** | | **Media** |
| RF 5 | Listar clientes ligeros. | Lista los clientes ligeros que han sido adicionados. |
| RF6 | Monitorear el estado de los clientes ligeros. | Permite conocer el estado Encendido/Apagado de los clientes ligeros. |
| RF7 | Mostrar el consumo de la RAM, la SWAP y la CPU de un cliente ligero | Permite al administrador el consumo de la RAM, la SWAP y la CPU de un cliente ligero. |
| RF8 | Finalizar los procesos ejecutados | Finaliza los procesos seleccionados previamente. |

| | | |
|---|---|---|
| | por el usuario. | |
| RF9 | Apagar clientes ligeros. | Apaga un cliente ligero. |

### 2.4.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos requisitos que no describen información a guardar, ni funciones a realizar, sino que son propiedades que hacen al producto usable, rápido o confiable. Además, se conocen como un conjunto de características de calidad, que es necesario tener en cuenta al diseñar e implementar el *software* (38).

**Restricciones del diseño e implementación**

**RnF1:** Se utilizará el lenguaje de programación Python y el *framework* Django.

**RnF2:** La interfaz visual debe mantener el estilo y diseño de la plataforma de administración de clientes ligeros Nova-LTSP.

**RnF3:** El sistema cumplirá con los patrones de diseño, Alta cohesión, Controlador y Decorador seleccionados por el equipo de desarrollo.

**Eficiencia**

**RnF4:** El sistema debe permitir que los usuarios interactúen con él de manera concurrente.

**RnF5:** El sistema debe ser capaz de responder 5000 peticiones en 5 segundos como máximo.

**Soporte**

**RnF6:** Se podrá acceder al sistema desde los navegadores web Mozilla Firefox en su versión 44 y Chrome en su versión 48.

**Seguridad**

**RnF7:** La seguridad de los datos se garantizará mediante conexiones seguras empleando el protocolo SSH.

***Hardware***

**RnF8:** El sistema se puede desplegar en un servidor que disponga de 128 Mb de memoria RAM o superior y 50 Mb de disco duro o superior.

## 2.5 Historias de Usuario

En la metodología AUP-UCI una de las formas que se utiliza para describir los requisitos de *software* son las Historias de Usuario (HU). Se describe brevemente y en un lenguaje natural, las características que el sistema debe poseer. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en un corto período de tiempo (39).

A continuación se describen las historias de usuarios de prioridad Alta definidas para el desarrollo de la propuesta de solución.

Tabla 2: HU Adicionar un cliente ligero.

| **Historia de Usuario** | |
|---|---|
| **Número:** HU_1 | **Nombre del requisito:** Adicionar un cliente ligero |
| **Programador:** Lexys Manuel Díaz Alonso | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 1 semana |
| **Riesgo en Desarrollo:** Alto | **Tiempo Real:** 1 semana |
| **Descripción:** La HU inicia cuando el usuario autenticado en la plataforma Nova-LTSP con privilegios de administración accede al menú Clientes y selecciona la opción: “Adicionar cliente ligero”, ubicado en el panel superior de la plataforma. El sistema le muestra al usuario un formulario para adicionar un cliente ligero con los siguientes campos:<br>**Nombre**(Campo obligatorio, campo de texto, este campo solo puede contener letras y números)<br>**Dirección MAC** (Campo obligatorio, campo de texto, dirección MAC válida)<br>**Dirección IP** (Campo obligatorio, campo de texto, dirección IP válida)<br>**Descripción** (Opcional, campo de texto, admite todos los caracteres)<br>El usuario introduce la información y presiona el botón: “Salvar”. El sistema verifica la información y envía un mensaje de confirmación, finalizando así la HU. | |
| **Observaciones:**<br>1. Si el usuario introduce la información de forma incorrecta, el sistema emite un mensaje notificando el error.<br>2. Si el usuario introduce la información dejando campos obligatorios vacíos, el sistema emite un mensaje indicándole que los campos obligatorios deben de llenarse. | |

**Prototipo elemental de interfaz gráfica de usuario:**


Tabla 3: HU Editar un cliente ligero.

| **Historia de Usuario** | |
|---|---|
| **Número:** HU_2 | **Nombre del requisito:** Editar un cliente ligero |
| **Programador:** Lexys Manuel Díaz Alonso | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 1 semana |
| **Riesgo en Desarrollo:** Alto | **Tiempo Real:** 1 semana |
| **Descripción:** La HU inicia cuando el usuario autenticado en la plataforma Nova-LTSP con privilegios de administración accede al menú Clientes. El sistema le muestra al usuario un listado con los clientes ligeros existentes en el sistema. El usuario selecciona el cliente a editar y luego va a la opción: “Editar Cliente Ligero”, ubicado en el panel superior de la plataforma. El sistema le muestra un formulario para editar el cliente ligero con los siguientes campos:<br>**Nombre**(Campo obligatorio, campo de texto, este campo solo puede contener letras y números)<br>**Dirección MAC** (Campo obligatorio, campo de texto, dirección MAC válida)<br>**Dirección IP** (Campo obligatorio, campo de texto, dirección IP válida)<br>**Descripción** (Opcional, campo de texto, admite todos los caracteres)<br>El usuario modifica la información y presiona el botón: “Salvar”. El sistema actualiza la información y envía un mensaje de confirmación, finalizando así la HU. | |
| **Observaciones:**<br>1. Si el usuario introduce la información de forma incorrecta, el sistema emite un mensaje notificando el error.<br>2. Si el usuario introduce la información dejando campos obligatorios vacíos, el sistema emite un mensaje | |

| |
|---|
| indicándole que los campos obligatorios deben de llenarse.<br>3. Si el usuario no selecciona el cliente ligero a editar, el sistema emite un mensaje notificando el error. |
| **Prototipo elemental de interfaz gráfica de usuario:**<br>  |

Tabla 4: HU Eliminar un cliente ligero.

| **Historia de Usuario** | |
|---|---|
| **Número:** HU_3 | **Nombre del requisito:** Eliminar un cliente ligero |
| **Programador:** Lexys Manuel Díaz Alonso | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 1 semana |
| **Riesgo en Desarrollo:** Alto | **Tiempo Real:** 1 semana |
| **Descripción:** La HU inicia cuando el usuario autenticado en la plataforma Nova-LTSP con privilegios de administración accede al menú Clientes. El sistema le muestra al usuario un listado con los clientes ligeros existentes en el sistema. El usuario selecciona el cliente a eliminar y luego va a la opción: “Eliminar Cliente Ligero” ubicado en el panel superior de la plataforma. El sistema muestra un mensaje de confirmación de la acción seguido de las opciones:<br>✓ Eliminar<br>✓ Cancelar<br>El usuario selecciona la opción “Cancelar”, el sistema deshace la operación y muestra el listado de los clientes ligeros existente en el sistema.<br>El usuario selecciona la opción “Eliminar”, el sistema elimina el cliente ligero y muestra un mensaje notificando la acción, finalizando así la HU. | |
| **Observaciones:**<br>1. Si el usuario no selecciona el cliente ligero a eliminar, el sistema emite un mensaje notificando el error. | |

**Prototipo elemental de interfaz gráfica de usuario:**


Tabla 5: HU Conectar con el escritorio de los clientes ligeros.

| **Historia de Usuario** | |
|---|---|
| **Número:** HU_5 | **Nombre del requisito:** Conectar con el escritorio de los clientes ligeros |
| **Programador:** Lexys Manuel Díaz Alonso | **Iteración Asignada:** 2 |
| **Prioridad:** Alta | **Tiempo Estimado:** 1 semana |
| **Riesgo en Desarrollo:** Alto | **Tiempo Real:** 1 semana |
| **Descripción:** La HU inicia cuando el usuario autenticado en la plataforma Nova-LTSP con privilegios de administración accede al menú Clientes. El sistema le muestra al usuario un listado con los clientes ligeros encendidos y apagados. El usuario selecciona un cliente encendido y elige la opción “Detalles del cliente ligero”, ubicado en el panel superior de la plataforma. El sistema le muestra al usuario la Última vista del escritorio con la opción de ver otras vistas del escritorio, finalizando así la HU. | |
| **Observaciones:** | |

**Prototipo elemental de interfaz gráfica de usuario:**


## 2.6 Planificación

En este punto se efectúa una planificación del sistema por etapas, donde se aplicarán diferentes iteraciones, y por cada iteración el cliente ha de recibir una versión nueva (40). A continuación, se define el plan de iteraciones que presidirán el proceso de desarrollo.

### 2.6.1 Plan de iteraciones

Una vez definidas las historias de usuario es conveniente proceder a la planificación de la fase de ejecución. El plan de entrega se compone de tres iteraciones, las cuales fueron fragmentadas para así obtener un trabajo incremental.

**Iteración 1:**

En la primera iteración se implementan las historias de usuario con mayor prioridad, obteniendo al final de la misma una primera versión de prueba y dando al sistema las primeras funcionalidades.

**Iteración 2**

En la segunda iteración se realiza la implementación de los requisitos correspondientes a las historias de usuarios con prioridad alta y media para el cliente, además se corrigen los errores encontrados en las historias de usuarios desarrolladas durante la primera iteración. Al concluir se obtiene la primera versión de la aplicación.

Tabla 6: Plan de iteraciones.

| Iteración | Descripción de la iteración | Orden de las HU a implementar | Semanas estimadas |
|---|---|---|---|
| 1 | Desarrollo de las historias de usuario de prioridad alta. | 1,2,3,4 | 5 semanas |
| 2 | Desarrollo de las historias de usuario de prioridad media. | 5,6,7,8,9 | 5 semanas |

## 2.7 Diseño del módulo

La actividad de diseño de *software* se refiere al establecimiento de las estructuras de datos y la arquitectura general del *software*, de manera que se satisfagan los requerimientos del *software*. El proceso de diseño traduce los requisitos en una representación de *software* (41).

A continuación, se define la arquitectura de *software* y los patrones de diseño que utiliza el módulo a implementar.

### 2.7.1 Arquitectura de software

La arquitectura de *software* es la estructura de un sistema o bien la forma en que va a estar organizado este; incluye los elementos que lo conforman, sus propiedades visibles desde el exterior y las relaciones que existen entre ellos (41).

La arquitectura que se utilizará para el desarrollo del módulo es la empleada por Django, el cual sigue una arquitectura Modelo-Vista-Controlador, solo que hace una adaptación de esta a Modelo-Vista-Plantilla (a partir de ahora MTV por sus siglas en inglés, Model Template View). Por tanto, el sistema propuesto hereda una arquitectura MTV, debido a que se desarrollará sobre el marco de trabajo Django.

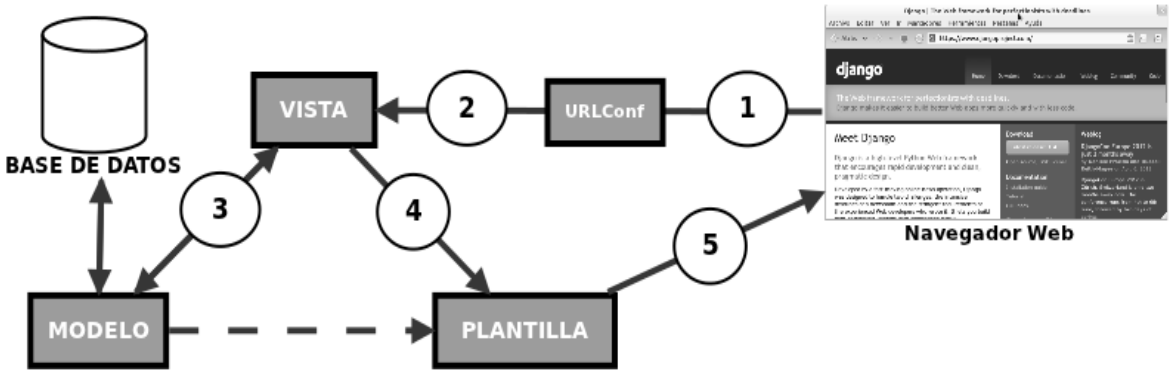



Figura 3 Funcionamiento del MTV de Django (32).

**Modelo:** Contiene toda la información sobre los datos. Cada una de las entidades de la base de datos se encuentra en el modelo en forma de clases de Python, y sus atributos se almacenan en variables con ciertos parámetros. También estos archivos poseen métodos, lo que permite indicar y controlar el comportamiento de los datos.

**Vista:** Es la capa de la lógica de negocios, contiene la lógica que accede al modelo y la delega a la plantilla apropiada. Esta capa sirve de "puente" entre el modelo y la plantilla, se presenta en forma de funciones de Python y su función principal es determinar qué datos serán visualizados en las plantillas.

**Plantilla:** Recibe los datos de la vista y luego los organiza para la presentación al navegador web. Básicamente es una página HTML (*HyperText Markup Language*) con algunas etiquetas extras que son propias del Django.

La Figura 8 representa la separación de las clases de la aplicación en el marco de trabajo según el patrón arquitectónico MVT.

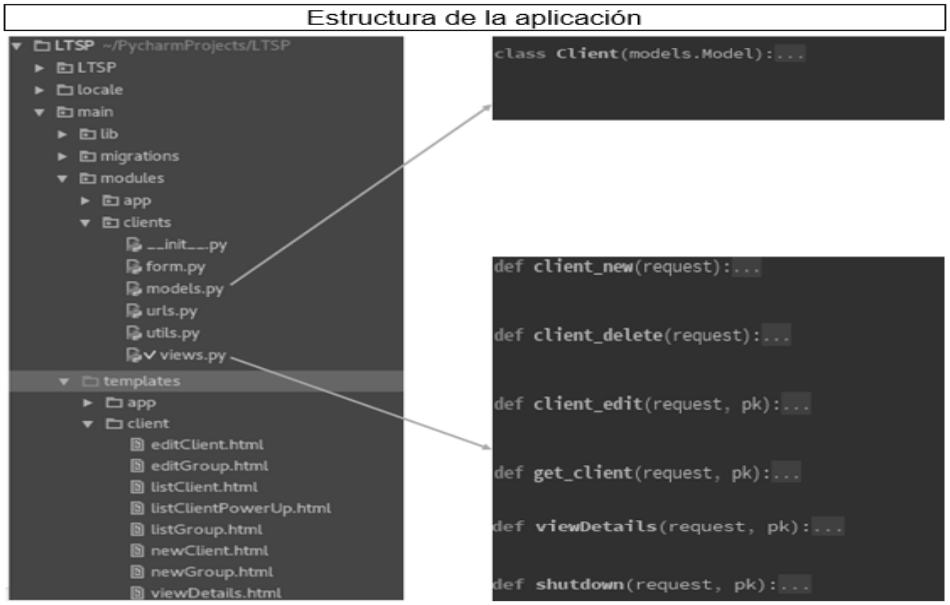



Figura 4: Arquitectura de la solución.

**Estructura de la aplicación**

**models.py:** Contiene la clase *Client,* responsable de brindar información de los clientes ligeros.

**views.py:** Contiene todas la *views* encargadas de acceder a la clase del *models.py* y llamar a las plantilas contenidas en la carpeta templates.

**templates:** Es la carpeta que contiene todas las plantillas utilizadas en la aplicación y llamadas desde las *views*.

### 2.7.2 Patrones de diseño

Un patrón de diseño es una buena práctica documentada de la solución de un problema que ha sido aplicado satisfactoriamente en múltiples entornos. Es una solución recurrente a un problema común observado o descubierto durante el estudio o construcción de numerosas aplicaciones. Su principal objetivo es incrementar la calidad del *software* en términos de reusabilidad, mantenimiento y extensibilidad (42).

**Patrones GRASP**

Los patrones GRASP (por sus siglas en inglés, *General Responsibility Assignment Software Patterns*) describen los principios fundamentales de la asignación de responsabilidades a objetos. El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el *software* orientado a objetos (42).

Los patrones GRASP utilizados en la solución propuesta fueron:

**Alta cohesión:** Asigna responsabilidades de manera tal que la cohesión siga siendo alta, o sea que las funcionalidades de las clases estén altamente relacionadas de forma tal que exista una colaboración entre ellas para compartir el esfuerzo y no caiga todo el peso sobre una única clase. Usar este patrón simplifica el mantenimiento y favorece el bajo acoplamiento.

```
def viewDetails(request, pk):
    client = get_object_or_404(Client, pk=pk)
    if not ClientUtils.pingToClient(ip=client.ip):
        notify.add(0, "El cliente ligero se encuentra apagado en estos momentos.")
        return redirect('listClient')

    def pingToClient(ip):
        response = os.system("ping -c 1 " + ip)
        if response == 0:
            return True
        else:
            return False
```

Figura 5: Código para saber si el cliente ligero se encuentra apagado.

Este patrón se evidencia en la propuesta de solución en la funcionalidad *viewDetails()* para conocer el consumo de los recursos de *hardware* de un cliente ligero en el servidor, dado que esta funcionalidad verifica primero que el cliente no se encuentra apagado, para ello hace uso de la funcionalidad *pingToClient()* ubicada en la clase Utils.

**Controlador:** Asigna la responsabilidad de gestionar un mensaje de un evento del sistema a una clase controladora. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

```
def client_edit(request, pk):
    client = get_object_or_404(Client, pk=pk)

    if request.method == "POST":
        form = ClientForm(request.POST, instance=client)
        if form.is_valid():
            client = form.save(commit=False)
            client.name = form.cleaned_data['name']
            client.mac = form.cleaned_data['mac']
            client.ip = form.cleaned_data['ip']
            client.description = form.cleaned_data['description']
            client.perfil = "default"
            client.hasImage = False
            client.save()
            notify.add(0, "El cliente " + form.cleaned_data['mac'] + " se ha editado con exito")
            return redirect('listClient')
    else:
        print client.mac
        form = ClientForm(instance=client)
    return render(request, 'client/editClient.html', {'form': form})
```

Figura 6: Código para editar un cliente ligero.

Este patrón se utiliza en la funcionalidad *client_edit()* para editar un cliente ligero. Dicha funcionalidad recibe los datos enviados por el usuario, los modifica en la base de datos, los renderiza a la plantilla *listClient* y el sistema emite un mensaje de confirmación notificando que el cliente ligero se ha editado con éxito.

**Patrones GoF**

Los patrones GoF (por sus siglas en inglés, *Gang of Four*) agrupan los patrones de acuerdo a su propósito: creación, estructura y comportamiento (41).

A continuación, se describe el patrón GoF utilizado en la solución propuesta:

**Decorador:** Patrón estructural que extiende la funcionalidad de un objeto dinámicamente de manera tal que es transparente a sus clientes, utiliza una instancia de una subclase de la clase original que delega las operaciones al objeto original.

```
@login_required(login_url='/')
def client_new(request):
    if request.method == "POST":
        form = ClientForm(request.POST)
        if form.is_valid():
            client = form.save(commit=False)
            client.name = form.cleaned_data['name']
            client.mac = form.cleaned_data['mac']
            client.ip = form.cleaned_data['ip']
            client.hasImage = False
            client.description = form.cleaned_data['description']
            client.perfil = "default"
            client.save()
            notify.add(0, "El cliente " + form.cleaned_data['mac'] + " se ha creado con exito.")
            return redirect('listClient')
    else:
        form = ClientForm()
    return render(request, 'client/newClient.html', {'form': form})
```

Figura 7: Código para crear un nuevo cliente ligero.

Como ejemplo de este patrón se evidencia la utilización del decorador de Django (@login_required) para garantizar que el usuario esté autenticado y tenga permiso para acceder a la funcionalidad *cliente_new()* para adicionar un nuevo cliente ligero.

### 2.8 Diagrama de clases del diseño

A diferencia del modelo de dominio, un diagrama de clases de diseño muestra las definiciones de las entidades del *software* en vez de conceptos del mundo real (42).

Los diagramas de clases del diseño utilizando estereotipos web especifican la estructura de clases de un sistema, así como sus relaciones. Definen de forma correcta las relaciones de dependencia, generalización y asociación de clases que constituyen el sistema.

**Los estereotipos usados son:**

<<Server page>> Representa la página web que contiene código que se ejecuta en el servidor.

<<Client Page>> Una instancia de Página Cliente es una página web, con formato HTML. Cada página cliente es construida por una sola página de servidor.

<<Form>> Colección de elementos de entrada que son parte de una página cliente. Sus atributos son los elementos de entrada del formulario.

<<Submit>> Es la relación que se crea siempre entre una página servidor y un formulario, a través de esta relación el formulario manda los valores de sus campos al servidor, para ser procesados por la página servidor.

<<Build>> Representa una asociación especial que relaciona las páginas cliente con las páginas servidor, de forma general se expresa como que las páginas que se encuentran en el servidor construyen las páginas en el cliente.

<<Redirect>> Indica que la página de servidor además de construir una página cliente puede redireccionar el procesamiento a otra página.

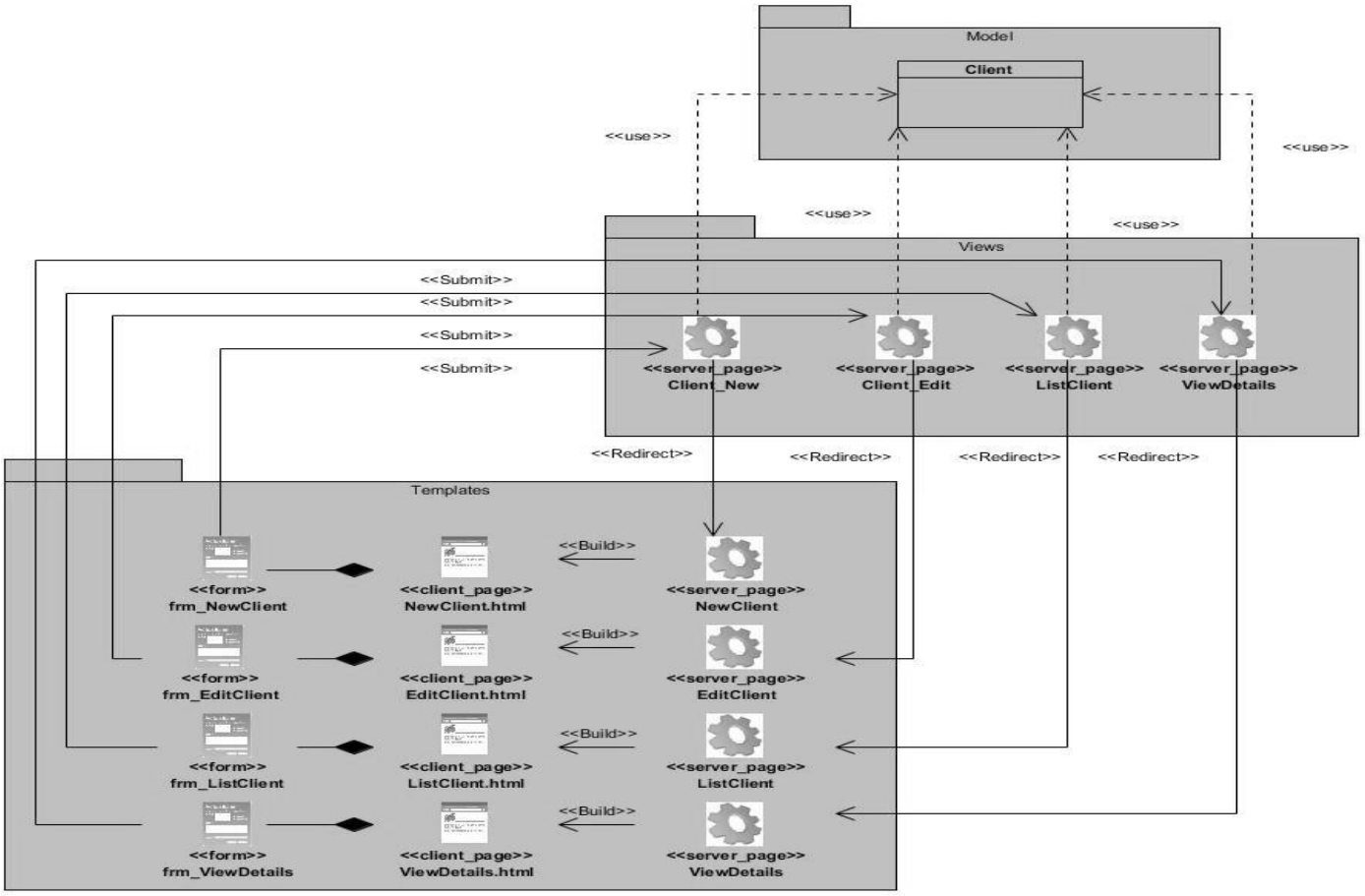



Figura 8: Diagrama de clases del diseño.

Para adicionar un cliente ligero la página servidora NewClient construye la página cliente NewClient.html, esta última se compone de un formulario denominado frm_NewClient donde se debe introducir los datos correspondientes que son ejecutados por la página servidora Client_New. Para editar un cliente ligero la página servidora EditClient construye la página cliente EditClient.html la cual tiene un formulario frm_EditClient en el cual se editan los datos y se ejecuta por la página servidora Client_Edit. Para listar los clientes ligeros la página servidora ListClient construye la página cliente ListClient.html la cual contiene un formulario frm_ListClient donde se listan los datos del cliente y se comunica con la página servidora ListClient. Para conocer los detalles de un cliente ligero la página servidora ViewDetails construye la página cliente ViewDetails.html, compuesta por un formulario llamado frm_ViewDetails en el cual se muestran los detalles del cliente ligero y se ejecutan por la página servidora ViewDetails.

## 2.9 Diagrama de paquetes

En el Lenguaje Unificado de Modelado, un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre ellas. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema (42).

Para el diseño del diagrama de paquetes se toma como referencia la arquitectura anteriormente propuesta. A continuación se presenta el diagrama de paquetes correspondiente al módulo a desarrollar.

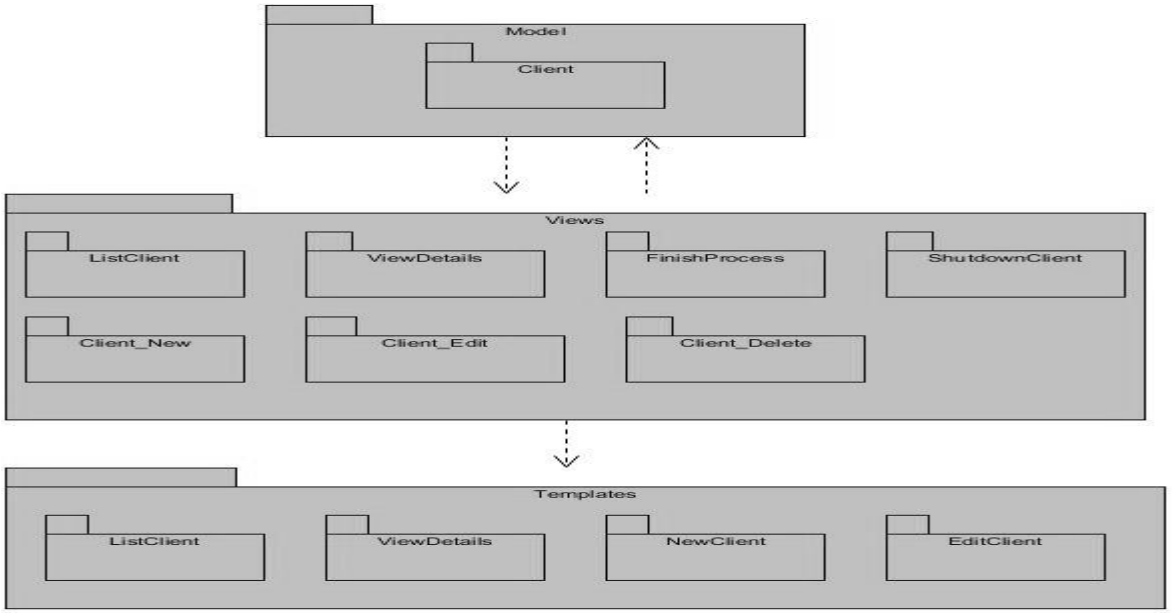



Figura 9: Diagrama de paquetes.

**Paquete Model**

**Client:** Contiene información sobre los clientes ligeros.

**Paquete Views**

**ListClient:** Vista que permite listar los clientes ligeros.

**ViewDetails:** Vista que permite conocer los clientes ligeros.

**FinishProcess:** Vista que permite finalizar los procesos que se ejecutan.

**ShutdownClient:** Vista que permite apagar un cliente ligero.

**Client_New:** Vista que permite adicionar un cliente ligero.

**Client_Edit:** Vista que permite editar un cliente ligero.

**Client_Delete:** Vista que permite eliminar un cliente ligero**.**

**Paquete Templates**

**ListClient:** Plantilla HTML que se encarga de listar los clientes ligeros.

**ViewDetails:** Plantilla HTML que se encarga de mostrar los detalles de un cliente ligero.

**NewClient:** Plantilla HTML que se encarga de capturar los datos para adicionar un cliente ligero.

**EditClient:** Plantilla HTML que se encarga de capturar los datos para editar un cliente ligero.

## 2.10 Modelo de despliegue

Un diagrama de despliegue modela la arquitectura de un sistema en tiempo de ejecución; muestra la configuración de los elementos de *hardware* (nodos) y muestra cómo los elementos y artefactos del *software* se trazan en esos nodos (43). Las relaciones que existen entre nodos representan los protocolos de comunicación que se utilizan para acceder a cada uno.

A continuación, se muestra el diagrama de despliegue definido para la solución propuesta. La PC Cliente representa las computadoras de los usuarios que se conectan al sistema, las cuales realizan peticiones al servidor LTSP mediante el protocolo HTTPS. Este servidor mantiene una conexión mediante el protocolo de comunicación segura SSH con los clientes ligeros, y mediante el protocolo TCP/IP al servidor de bases de datos.

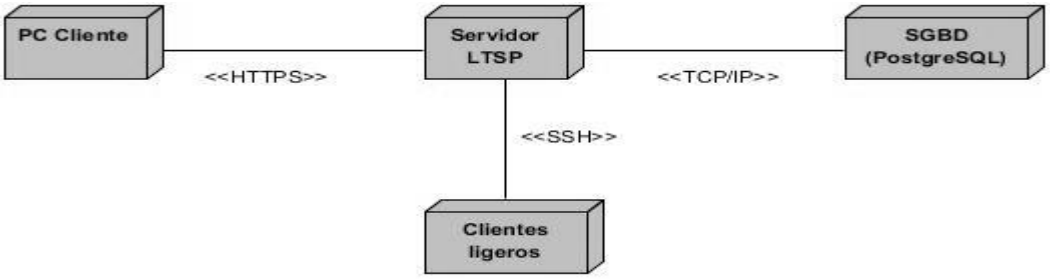



Figura 10: Diagrama de despliegue.

## 2.11 Conclusiones parciales

En este capítulo se han abordado los elementos del análisis y diseño del módulo de monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, arribando a las siguientes conclusiones:

- ✓ Los requisitos funcionales y no funcionales obtenidos a partir del proceso de identificación de los requisitos, sirvieron de guía para desarrollar las distintas funcionalidades y de este modo satisfacer las necesidades detectadas.
- ✓ La utilización de los patrones de diseño permitió identificar aspectos importantes de la estructura del diseño del sistema propuesto, lo que garantizó una mayor organización e hizo el código más legible.
- ✓ Las descripciones de las historias de usuario y la elaboración de los diagramas de clases del diseño y el diagrama de paquetes posibilitaron una mejor comprensión del funcionamiento de la propuesta de solución.
- ✓ Adoptar la arquitectura de *software* Modelo-Vista-Plantilla propuesta por el *framework* de desarrollo utilizado, permitió una propicia organización del sistema a implementar.

## Capítulo 3. Implementación y pruebas al módulo

En el presente capítulo se muestran los artefactos correspondientes a las etapas de implementación y prueba del sistema, así como los estándares de codificación que debe seguir el equipo de desarrollo para implementar el *software*. De acuerdo a la metodología que se utiliza, se especifican, de los tipos de pruebas que esta plantea, los que serán empleados para darle validez a los requisitos funcionales y garantizar el óptimo funcionamiento de la aplicación. Además, se realiza una valoración de los resultados obtenidos al aplicar las pruebas.

La implementación del sistema es una de las fases imprescindibles dentro del proceso de desarrollo de *software*. Esta fase comprende la materialización, en forma de código, de todos los artefactos, descripciones y arquitectura propuestos en la etapa de análisis y diseño; con el objetivo de conformar el producto final requerido por el cliente (42).

### 3.1 Modelo de implementación

El modelo de implementación describe como los elementos del modelo de diseño, cómo las clases, se implementan en términos de componentes, como fichero de código fuente y ejecutables. El modelo implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros (42).

**Diagrama de componentes**

Los diagramas de componentes son utilizados para estructurar el modelo de la implementación. Permiten modelar una vista estática del sistema, muestran la organización y las dependencias lógicas entre un conjunto de componentes del *software*, que pueden ser librerías, binarios, ejecutables y códigos fuentes (42).

**Descripción de los componentes del sistema**

**Plantilla:** Paquete que agrupa a las plantillas encargadas de presentar la información al usuario.

**Vista:** Paquete que agrupa a todos los componentes que interactúan con el paquete de clases Modelo; estos componentes permiten trabajar con algunas utilidades sobre los formularios y la renderización de la información en las plantillas apropiadas.

**Modelo:** Paquete que agrupa las clases que representan el dominio de entidades de la base de datos y que permiten la interacción directa con el paquete Vista.

**Django ORM:** Componente encargado de mapear, crear, acceder y modificar la información que se encuentra en la base de datos.

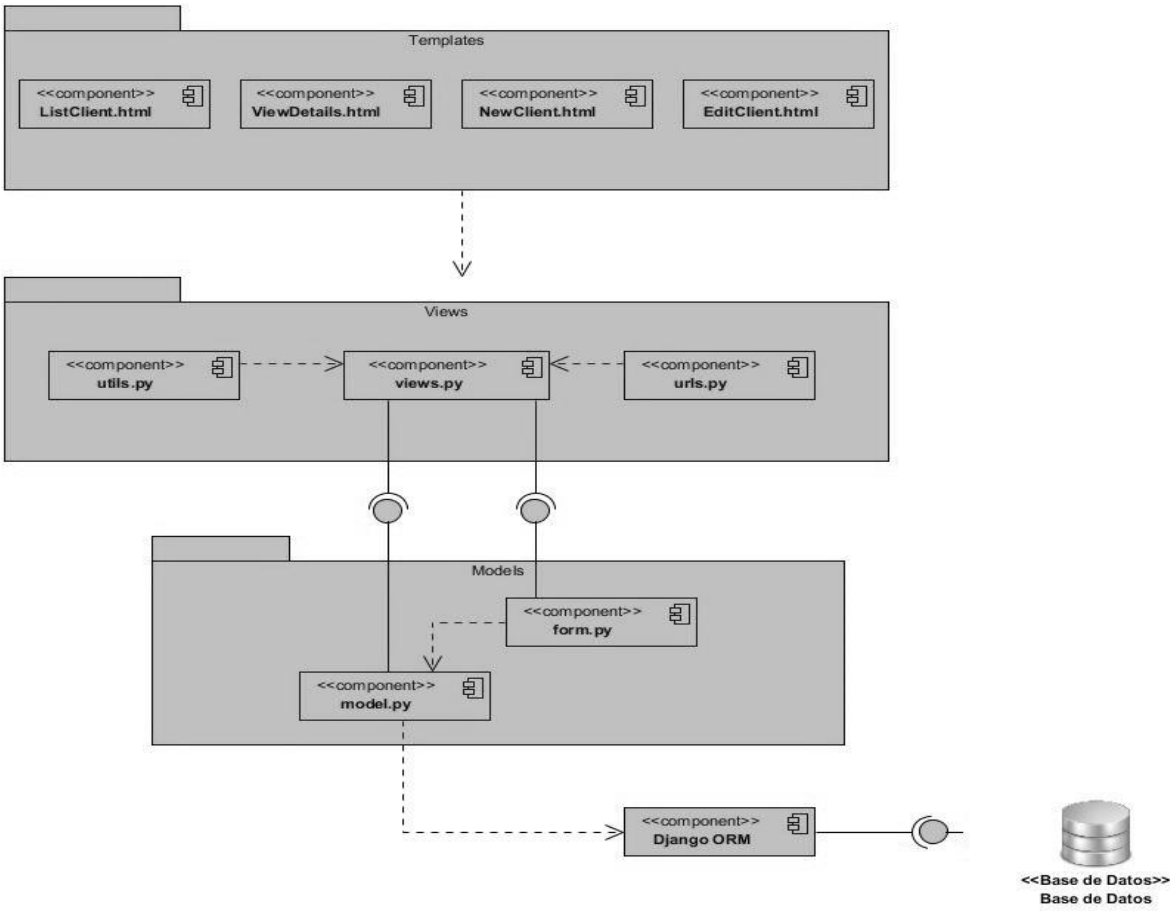



Figura 11: Diagrama de componentes.

## 3.2 Estándares de codificación

Cada programador tiene su propia formar de escribir los códigos y puede ser completamente diferente a la de otros programadores, pero de la forma que se use depende la facilidad de que otros programadores entiendan el código y se les facilite su reutilización, de ahí se desprende la importancia de los estilos de programación, también conocidos como estándares o convenciones de código los cuales definen un grupo de convenciones para escribir código fuente en ciertos lenguajes de programación.

Para la codificación de la propuesta de solución se utilizará el estándar de codificación para Python basado en la guía de estilo del código Python por Guido Van Rossum, Barry Warsaw. En este documento se listan distintas convenciones utilizadas en el código Python comprendido en la librería estándar de la distribución principal de Python (44).

**Identación**

- ✓ Usa cuatro espacios por cada nivel de identación.
- ✓ Las líneas de continuación deben alinearse verticalmente con el carácter que se ha utilizado (paréntesis, llaves, corchetes).
- ✓ No se mezclarán tabuladores y espacios en la codificación. El método de identación más popular en Python es con espacios. El segundo más populares con tabulaciones, sin mezclar unos con otros. Cualquier código identado con una mezcla de espacios y tabulaciones debe ser convertido a espacios exclusivamente.

**Máxima longitud de las líneas**

- ✓ Se limitarán todas las líneas a un máximo de 79 caracteres.
- ✓ Dentro de paréntesis, corchetes o llaves se puede utilizar la continuación implícita para cortar las líneas largas.
- ✓ En cualquier circunstancia se puede utilizar el carácter “\” para cortar las líneas largas.

**Líneas en blanco**

- ✓ Separar las funciones de alto nivel y definiciones de clases con dos líneas en blanco.
- ✓ Las definiciones de métodos dentro de una clase deben separarse por una línea en blanco.
- ✓ Se puede utilizar líneas en blanco escasamente para separar secciones lógicas.

**Codificaciones**

- ✓ Utilizar la codificación UTF-8.
- ✓ Se pueden incluir cadenas que no correspondan a esta codificación utilizando “\x”, “\u” o “\U”.

**Importaciones**

- ✓ Las importaciones deben estar en líneas separadas.
- ✓ Las importaciones siempre deben colocarse al comienzo del archivo.
- ✓ Las importaciones deben estar agrupadas de la siguiente forma:
- Importaciones de la librería estándar.
- Importaciones terceras relacionadas.
- Importaciones locales de la aplicación / librerías.
- ✓ Cada grupo de importaciones debe estar separado por una línea en blanco.

**Espacios en blanco en expresiones y sentencias**

- ✓ Evitar utilizar 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.
- Inmediatamente antes del paréntesis que comienza la lista de argumentos en la llamada a una función.
- Inmediatamente antes de un corchete que empieza una indexación.
- Más de un espacio alrededor de un operador de asignación (u otro) para alinearlo con otro.
- ✓ Deben rodearse con exactamente un espacio los siguientes operadores binarios:
- Asignación (=).
- Asignación de aumentación (+=, -=).
- Comparación (==, <, >, >=, <=, ! =, <>, *in, not in, is, is not*).
- Expresiones lógicas (*and, or, not*).
- ✓ Si se utilizan operadores con prioridad diferente se aconseja rodear con espacios a los operadores de menor prioridad.

- ✓ No utilizar espacios alrededor del igual (=) cuando es utilizado para indicar un argumento de una función o un parámetro con un valor por defecto.

**Comentarios**

- ✓ Los comentarios deben ser oraciones completas.
- ✓ Si un comentario es una frase u oración su primera palabra debe comenzar con mayúscula a menos que sea un identificador que comience con minúscula.
- ✓ Nunca cambiar las minúsculas y mayúsculas en los identificadores de clases, objetos y funciones.
- ✓ Si un comentario es corto el punto final puede omitirse.

**Comentarios en bloque**

- ✓ Deben estar indentados al mismo nivel que el código a comentar.
- ✓ Cada línea de un comentario en bloque comienza con un numeral (#) y un espacio en blanco.

**Comentarios en línea**

- ✓ Se recomienda utilizarlos escasamente.
- ✓ Se debe definir comenzando por un numeral (#) seguido de un espacio en blanco.
- ✓ Deben ubicarse en la misma línea que se desea comentar.

**Convenciones de nombramientos**

- ✓ Nunca se deben utilizar como simple caracteres para nombres de variables los caracteres ele minúscula “l”, o mayúscula “O”, ele mayúscula “L” ya que en algunas fuentes son indistinguibles de los números uno y cero.
- ✓ Los módulos deben tener un nombre corto y en minúscula.
- ✓ Los nombres de clases deben utilizar la convención “CapWords” (palabras que comienzan con mayúsculas).
- ✓ Los nombres de las excepciones deben estar escrito también en la convención “CapWords” utilizando el sufijo “Error”.

- ✓ Los nombres de las funciones deben estar escrito en minúscula separando las palabras con un guión bajo "_".
- ✓ Las constantes deben quedar escritas con letras mayúsculas separando las palabras por un guión bajo (_).

## 3.3 Pruebas de software

Las pruebas de *software* son un elemento crítico para la garantía de calidad del *software*, y representan una revisión final de las especificaciones del diseño y de la codificación. Son utilizadas para identificar posibles fallos de implementación, calidad o usabilidad de un programa (21).

La metodología que guía el proceso de desarrollo de la presente investigación propone realizar pruebas internas, de liberación y de aceptación. Las pruebas de liberación son diseñadas y ejecutadas por una entidad certificadora externa, por lo que en el presente trabajo no son analizadas.

### 3.3.1 Estrategias de prueba

Una estrategia de prueba del *software* integra los métodos de diseño de casos de prueba en una serie bien planteada de pasos que va a converger en la eficaz construcción del *software*. Dicha estrategia debe ser lo suficientemente flexible como para promover un enfoque personalizado, y al mismo tiempo lo adecuadamente rígido como para originar una planeación razonable y un seguimiento administrativo del avance del producto (45).

Pressman plantea las siguientes estrategias de prueba:

- ✓ Pruebas de unidad
- ✓ Pruebas de integración
- ✓ Pruebas de validación
- ✓ Pruebas de sistema

Las pruebas de unidad se centran en cada unidad del *software*, tal como se implementó en el código fuente. La prueba de integración se enfoca al diseño y la construcción de la arquitectura del *software*. La prueba de validación permite validar los requisitos establecidos como parte del análisis de requisitos de

*software*, comparándolos con el sistema que ha sido construido. Finalmente se prueba el *software* como un todo empleando la prueba del sistema (45).

A la solución que se propone se le realizará cada una de las pruebas antes definidas.

#### 3.3.1.1 Pruebas de unidad

El objetivo de estas pruebas es ejecutar un código fuente al llamar directamente a los métodos de una clase pasándole a estos los parámetros adecuados.

Las pruebas de unidad descomponen las funciones del programa en comportamientos comprobables discretos que se pueden probar como unidades individuales. Están destinadas a verificar las unidades más pequeñas del s*oftware*. Se aplican a las funcionalidades para verificar que los flujos de control y datos están cubiertos y funcionan tal como se espera. La prueba de unidad siempre está orientada a caja blanca (45).

La prueba de caja blanca se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. Mediante la prueba de la caja blanca el ingeniero del *software* puede obtener casos de prueba que (45):

- ✓ Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- ✓ Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- ✓ Ejecuten todos los bucles en sus límites operacionales.
- ✓ Ejerciten las estructuras internas de datos para asegurar su validez.

La prueba de unidad que se realiza hace uso del método de caja blanca y de la técnica del camino básico. El método del camino básico permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño procedural y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

A continuación se realiza la técnica del camino básico a la funcionalidad Finalizar procesos que permite finalizar los procesos ejecutados por el usuario.

Tabla 7: Técnica del camino básico a la funcionalidad Finalizar procesos.

| | |
|---|---|
| **(1)** | def finishProcess(request, id): |
| **(2)** | client = get_object_or_404(Client, pk-id) |
| **(3)** | seleccionados = request.POST.getlist('processSelected') |
| **(4)** | s = Connection (str(client.ip), 'lexys' ,password='Lexys') |
| **(5)** | for pid in seleccionados: |
| **(6)** | s.execute ('skill' + pid) |
| **(7)** | notify.add (0, “Se finalizaron” + str (len(seleccionados)) + 'procesos del sistema' + client.ip + ' . ') |
| **(8)** | return HttpResponseRedirect('/cliente/viewDetails/' + str(id)) |

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

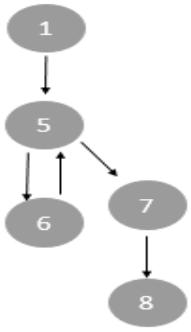



Figura 12: Grafo de flujo

A partir del grafo obtenido con 5 nodos y 5 aristas se calcula la complejidad ciclomática V(G), la cual constituye una métrica de *software* que proporciona una medida cuantitativa de la complejidad lógica del programa (45).

V(G)= (cantidad_aristas – cantidad_nodos) + 2

V(G)= (5 – 5) + 2 = 2

Una ruta independiente es cualquier ruta del programa que ingrese al menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición (45). La cantidad de rutas independientes son

establecidas por la complejidad ciclomática, por tanto se identifican dos rutas, tal y como se muestra en la siguiente tabla.

Tabla 8: Listado de caminos básicos.

| No Ruta | Caminos |
|---|---|
| 1. | 1-5-7-8 |
| 2. | 1-5-6-7-8 |

El valor de V(G) ofrece además un límite superior del número de pruebas que debe diseñarse y ejecutarse para garantizar la cobertura de todas las instrucciones (45). Por tanto, se diseñan casos de pruebas para ser aplicados a cada ruta independiente.

Tabla 9: Caso de prueba para la primera ruta.

| **Caso de Prueba de Unidad** | |
|---|---|
| **No. ruta:** 1 | **Ruta:** 1-5-7-8 |
| **Nombre de la persona que realiza la prueba:** Lexys Manuel Díaz Alonso | |
| **Descripción de la prueba**: Finalizar los procesos ejecutados por el usuario. | |
| **Entrada:** Se envía como parámetro el id del cliente, del cual se quiere finalizar los procesos. | |
| **Resultado esperado:** El sistema envía una notificación | |
| **Evaluación de la Prueba:** Satisfactoria. Se obtuvo el mensaje esperado. | |

Tabla 10: Caso de prueba para la segunda ruta.

| **Caso de Prueba de Unidad** | |
|---|---|
| **No. ruta:** 2 | **Ruta:** 1-5-6-7-8 |
| **Nombre de la persona que realiza la prueba:** Lexys Manuel Díaz Alonso | |
| **Descripción de la prueba**: Finalizar los procesos ejecutados por el usuario. | |
| **Entrada:** Se envía como parámetro el id del cliente, del cual se quiere finalizar los procesos. | |
| **Resultado esperado:** El sistema debe recorrer la lista de los procesos seleccionados y deben ser finalizados los procesos en el cliente. | |
| **Evaluación de la Prueba:** Satisfactoria. Se finalizaron todos los procesos. | |

**Resultados de las pruebas de unidad**

Se realizaron tres iteraciones de la prueba unitaria. En la primera iteración se detectaron 4 no conformidades; en la segunda, 1 no conformidad la cual fue resuelta para la tercera iteración. Las no conformidades detectadas estaban asociadas a errores de validación.

#### 3.3.1.2 Pruebas de integración

La prueba de integración es una técnica para construir la arquitectura del *software*. El objetivo es tomar componentes a los que se aplicó una prueba de unidad y construir una estructura de programa que determine el diseño (45). Asumiendo que el módulo debe incorporarse a una herramienta base, se emplea una estrategia de integración ascendente, donde los componentes se integran de abajo hacia arriba.

En el contexto de la prueba de integración ascendente se realiza la prueba de regresión que permite ejecutar nuevamente el mismo subconjunto de pruebas que ya se ha aplicado para asegurar que los cambios no han propagado efectos colaterales indeseables (45).

**Resultados de las pruebas de integración**

Las pruebas de integración realizadas permitieron identificar una URL duplicada en diferentes módulos de la plataforma Nova-LTSP. Luego de solucionar esta no conformidad, el módulo de monitoreo de los clientes ligeros desarrollado se integró correctamente con dicha plataforma.

#### 3.3.1.3 Pruebas de validación

Las pruebas de validación consisten en realizar acciones visibles para el usuario y en la salida que el sistema tiene ante cada una de estas acciones. La validación del *software* se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos funcionales. Las pruebas de caja negra se realizan sobre la interfaz del *software*, sin tener en cuenta el funcionamiento interno del sistema, por lo que los casos de prueba pretenden demostrar que las funciones del *software* son operativas (45).

A continuación se presentan los casos de prueba correspondientes a las historias de usuario Adicionar cliente ligero, Editar cliente ligero, Eliminar cliente ligero y Listar cliente ligero a partir de los cuales el cliente ejecutó las pruebas.

Tabla 11: Caso de prueba Adicionar cliente ligero.

| **Caso de Prueba de Aceptación** | |
|---|---|
| **Código Caso de Prueba:** Nova-LTSP-1-1 | **Nombre Historia de Usuario:** Adicionar cliente ligero |
| **Nombre de la persona que realiza la prueba:** Yasiel Pérez Villazón | |
| **Descripción de la prueba**: El sistema permitirá adicionar un cliente ligero insertando determinadas características. | |
| **Condiciones de Ejecución**: El usuario debe estar autenticado en la plataforma Nova-LTSP. Debe adicionar un cliente ligero. | |
| **Entrada/Pasos de Ejecución:**<br>1.Acceder al módulo Cliente<br>2.Seleccionar la opción Adicionar cliente ligero.<br>3.Llenar los campos.<br>4.Guardar los cambios. | |
| **Resultado esperado:** El sistema adiciona un nuevo cliente ligero, y muestra una notificación de creación exitosa. | |
| **Evaluación de la Prueba:** Satisfactoria. | |

Tabla 12: Caso de prueba Editar cliente ligero.

| **Caso de Prueba de Aceptación** | |
|---|---|
| **Código Caso de Prueba:** Nova-LTSP-2-2 | **Nombre Historia de Usuario:** Editar cliente ligero |
| **Nombre de la persona que realiza la prueba:** Yasiel Pérez Villazón | |
| **Descripción de la prueba**: el sistema permitirá editar los parámetros de un cliente ligero. | |
| **Condiciones de Ejecución**: El usuario debe estar autenticado en la plataforma Nova-LTSP. Debe editar un cliente ligero. | |
| **Entrada/Pasos de Ejecución:**<br>1.Acceder al módulo Cliente.<br>2.Seleccionar la opción Editar cliente ligero.<br>3.Editar los campos.<br>4.Guardar los cambios. | |
| **Resultado esperado:** El sistema editar un nuevo cliente ligero, y muestra una notificación de modificación exitosa. | |
| **Evaluación de la Prueba:** Satisfactoria. | |

Tabla 13: Caso de prueba Eliminar cliente ligero.

| **Caso de Prueba de Aceptación** | |
|---|---|
| **Código Caso de Prueba:** Nova-LTSP-3-3 | **Nombre Historia de Usuario:** Eliminar cliente ligero |
| **Nombre de la persona que realiza la prueba:** Yasiel Pérez Villazón | |
| **Descripción de la prueba**: El sistema permitirá eliminar un cliente ligero. | |
| **Condiciones de Ejecución**: El usuario debe estar autenticado en la plataforma Nova-LTSP. Debe eliminar un cliente ligero. | |
| **Entrada/Pasos de Ejecución:**<br>1.Acceder al módulo Cliente.<br>2.Seleccionar el cliente ligero a eliminar.<br>3.Seleccionar la opción Eliminar cliente ligero. | |
| **Resultado esperado:** El sistema elimina el cliente ligero y muestra una notificación de eliminación exitosa. | |
| **Evaluación de la Prueba:** Satisfactoria. | |

Tabla 14: Caso de prueba Listar los clientes ligeros.

| **Caso de Prueba de Aceptación** | |
|---|---|
| **Código Caso de Prueba:** Nova-LTSP-6-4 | **Nombre Historia de Usuario:** Listar los clientes ligeros |
| **Nombre de la persona que realiza la prueba:** Yasiel Pérez Villazón | |
| **Descripción de la prueba**: El sistema permitirá listar los clientes ligeros. | |
| **Condiciones de Ejecución**: El usuario debe estar autenticado en la plataforma Nova-LTSP. Debe listar los clientes ligeros. | |
| **Entrada/Pasos de Ejecución:**<br>1.Acceder al módulo Cliente. | |
| **Resultado esperado:** El sistema lista todos los clientes ligeros. | |
| **Evaluación de la Prueba:** Satisfactoria. | |

### Resultados de las pruebas de validación

Las pruebas de validación se ejecutaron tras concluir cada una de las tres iteraciones de implementación, de manera general en la primera iteración se detectaron 13 no conformidades, agrupadas en no conformidades de código, interfaz y ortografía, de ellas fueron solucionadas 9. En la segunda se detectaron 8 no conformidades incluyendo las 4 no resueltas en la iteración anterior, aquí se resolvieron 7 de ellas. Para una tercera iteración, solo se encontró la no conformidad que quedó pendiente en la iteración anterior, la misma se solucionó satisfactoriamente (Ver Figura 17).

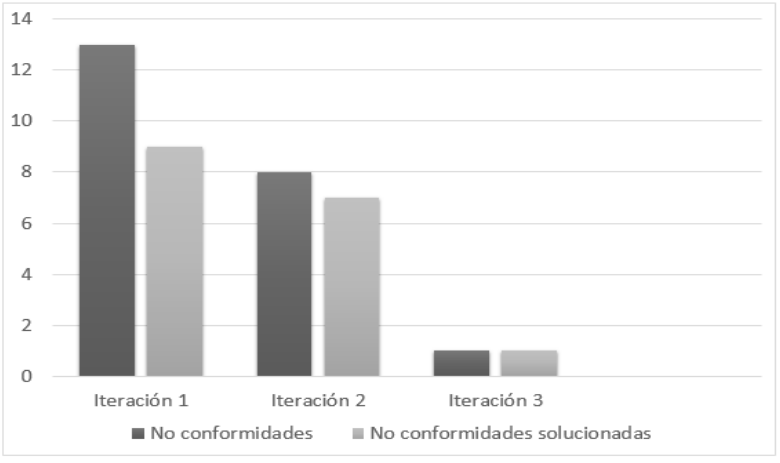



Figura 13: Resultados de las pruebas de validación.

#### 3.3.1.4 Pruebas del sistema

Las pruebas de sistema engloban una serie de pruebas cuyo propósito fundamental es ejercitar el sistema de cómputo. Cada una de estas pruebas tiene un objetivo distinto, pero todas trabajan para verificar que se hayan integrado adecuadamente todos los elementos del sistema y que realizan las funciones correctas (45). El tipo de prueba de sistema aplicada a la solución fue la prueba de rendimiento.

Las pruebas de rendimiento (también conocidas como pruebas de carga y estrés) tienen que diseñarse para asegurar que el sistema pueda procesar la carga esperada. Esto normalmente implica planificar una serie de pruebas en las que la carga se va incrementando regularmente hasta que el rendimiento del sistema se haga inaceptable (46).

Para la realización de las pruebas de carga y estrés se utilizó la herramienta Apache Jmeter en su versión 2.11. Es una aplicación de código abierto diseñada para medir el rendimiento de las aplicaciones a partir de comportamientos funcionales. Puede ser utilizado para probar recursos estáticos y dinámicos, servicios web, simular una carga pesada en un servidor, grupo de servidores, en la red y para hacer un análisis gráfico de rendimiento (47). Las pruebas se realizaron desde una computadora con 4GB de RAM, microprocesador Intel Core i3 con 3.30 GHz y sistema operativo Ubuntu 16.04. A continuación, se describen las variables que miden el resultado de las pruebas de carga y estrés realizadas al sistema.

**Muestra:** Cantidad de peticiones realizadas para cada URL.

**Media:** Tiempo promedio en milisegundos en el que se obtienen los resultados.

**Mediana:** Tiempo en milisegundos en el que se obtuvo el resultado que ocupa la posición central.

**Min:** Tiempo mínimo que demora un hilo en acceder a una página.

**Max:** Tiempo máximo que demora un hilo en acceder a una página.

**Línea 90%:** Máximo tiempo utilizado por el 90% de la muestra, al resto de la misma le llevó más tiempo.

**% Error:** Por ciento de error de las páginas que no se llegaron a cargar de manera satisfactoria.

**Rendimiento (Rend):** El rendimiento se mide en cantidad de solicitudes por segundo.

**KB/s:** El rendimiento se mide en cantidad de kilobytes[24] 10 por segundo.

[24] Kilobytes: Unidad de almacenamiento de información equivalente a 1024 bytes.

Debido a que el módulo está integrado a una plataforma de administración no es necesario establecer una cantidad elevada de usuarios conectados concurrentemente al sistema, no obstante se tomó como referencia un total de 100, 500 y 1000 usuarios con la finalidad de medir el rendimiento de la aplicación y asegurar que esta pueda procesar la carga esperada.

Como se muestra en la siguiente tabla, se simularon las peticiones realizadas al sistema por un total de 100, 500 y 1000 usuarios simultáneamente, obteniéndose los siguientes resultados:

Tabla 15: Resultados obtenidos a partir de las pruebas de carga y estrés.

| Usuarios | Muestra | Media | Mediana | Línea 90% | Min. | Max. | %Error | Rend. | KB/s |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 8250 | 1260 | 1397 | 1724 | 1 | 2666 | 0 | 75.7 | 655.3 |
| 500 | 42000 | 4170 | 2113 | 3110 | 5 | 321789 | 0.48 | 73.3 | 663.7 |
| 1000 | 68399 | 9672 | 2260 | 5142 | 2 | 506901 | 2.09 | 44.2 | 346.7 |

Las pruebas realizadas muestran que el sistema es capaz de responder a 8250 peticiones de 100 usuarios conectados simultáneamente en un tiempo promedio de 1260 milisegundos (1.3 segundos aproximadamente) con 0% de error. Esto evidencia que el sistema puede procesar la carga esperada, cumpliéndose de este modo el requisito no funcional 5.

Por otra parte, se realizaron 42000 peticiones iniciadas por 500 usuarios y en este caso el sistema respondió en 4170 milisegundos (4.2 segundos aproximadamente) como tiempo promedio. Esto demuestra que el sistema responde en el tiempo esperado a un conjunto de peticiones 8.4 veces mayor que el propuesto en el requisito no funcional 5, aunque no fue capaz de responder correctamente el 0.48% de las peticiones realizadas.

Por último, y con el objetivo de analizar el comportamiento del sistema en condiciones extremas, se realizó una prueba de estrés para un conjunto de 1000 usuarios conectados simultáneamente. En este caso, el sistema no responde adecuadamente, siendo capaz de responder solamente a 68399 peticiones de las 82500 esperadas, en aproximadamente el doble del tiempo propuesto en el requisito no funcional 5.

## 3.4 Validación de la hipótesis de investigación

Para verificar que, con la implementación y uso del módulo de monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, se favorece el control de los recursos de *hardware* en el servidor, se emplea el método criterio de expertos en su variante Delphi (48). El método consiste en la selección de un grupo de expertos a los que se les pregunta su opinión sobre cuestiones referidas a un problema definido, clasifica como uno de los métodos generales de prospectiva, que busca acercarse al consenso de un grupo de expertos con base en el análisis y la reflexión de dicho problema (49). Para la aplicación del método se emplearon las siguientes etapas:

**Selección de expertos**

Según las fuentes referenciadas, un experto es una persona, grupo de personas u organización con conocimientos amplios en un área particular del conocimiento, capaces de valorar, formular conclusiones objetivas y dar recomendaciones acerca del problema en cuestión.

En el caso de esta investigación, los expertos se seleccionaron teniendo en cuenta que cumplieran con los criterios siguientes:

- ✓ Experiencia profesional en administración de servidores.
- ✓ Nivel de dominio sobre el tema de clientes ligeros.
- ✓ Control y monitoreo de redes.

En la siguiente tabla se muestran los expertos seleccionados.

Tabla 16: Datos de los expertos seleccionados.

| Experto | Categoría | Años de experiencia |
|---|---|---|
| 1 | Máster | cuatro años |
| 2 | Ingeniero | cuatro años |
| 3 | Ingeniero | tres años |
| 4 | Ingeniero | dos años |
| 5 | Ingeniero | tres años |
| 6 | Ingeniero | cuatro años |
| 7 | Ingeniero | cuatro años |
| 8 | Ingeniero | tres años |

**Elaboración del cuestionario**

Una vez seleccionados los expertos se elaboró el cuestionario con cinco preguntas para validar la satisfacción de los usuarios y el cumplimiento de los requisitos del sistema. Cada pregunta se corresponde con los criterios a evaluar definidos por el investigador. Se incluye además, una última pregunta donde se solicita a los expertos sus recomendaciones para futuras versiones de la solución. Para evaluar el cuestionario se utiliza una escala descendente de 5 a 1: Muy adecuado (5), Bastante adecuado (4), Adecuado (3), Poco adecuado (2) e Inadecuado (1); siendo Muy adecuado el máximo valor posible.

**Establecimiento de la concordancia de los expertos mediante el coeficiente de Kendall**

Un acuerdo entre los expertos brinda mayor validez a la propuesta y para la determinación del consenso se calcula el Coeficiente de Concordancia de Kendall (W) que ayuda a comprobar el grado de coincidencia de las valoraciones realizadas por los expertos (50).

Tabla 17: Valores asignados por los expertos a cada criterio.

| Criterios/Expertos | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Rj |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 5 | 5 | 5 | 5 | 5 | 4 | 5 | 5 | 39 |
| **2** | 5 | 4 | 5 | 4 | 5 | 4 | 5 | 5 | 37 |
| **3** | 5 | 4 | 5 | 5 | 5 | 5 | 4 | 5 | 38 |
| **4** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 40 |
| **5** | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 40 |

Con los datos anteriores se tiene que:

K es el número de expertos que intervienen en el proceso de validación. En este caso K = 8.

N es cantidad de aspectos a validar. En este caso N = 5.

Rj es la suma de los rangos asignados a cada pregunta por parte de los expertos.

**(1)** $\overline{Rj} = \sum_{1}^{8} Rj/N$ **(2)** $S = \sum_{1}^{8} (Rj - \overline{Rj})^2$ **(3)** $W = 12S / K^2(N^3 - N)$ **(4)** $X^2 \text{ real} = K(N-1)W$

Figura 14: Fórmulas utilizadas en el método criterio de expertos.

Rj media, es la media de los rangos y se determina mediante la fórmula **(1)**, obteniendo el valor: 38.8.

S es la suma de los cuadrados de las desviaciones y se calcula mediante la fórmula **(2)**, donde S = 6.8; W es el coeficiente de Kendall y se calcula con la fórmula **(3)**.

Sustituyendo los valores obtenidos en la ecuación, W = 0.01. El coeficiente W ofrece el valor que posibilita decidir el nivel de concordancia entre los expertos. El valor de W siempre es positivo y oscila entre 0 y 1.

Con el coeficiente de Kendall se puede calcular el Chi-Cuadrado real con el objetivo de ver si existe o no concordancia entre los expertos, el mismo se obtiene a través de la fórmula **(4)**. Obteniendo como Chi-Cuadrado real el valor: 0.32.

Este Chi-Cuadrado calculado se compara con el tabulado en la tabla del percentil de la distribución de la variable Chi-Cuadrado. Para tener un 95% de confianza se utilizará α =0.05. Si el Chi-Cuadrado real ($\chi 2$ real) es menor que el Chi-Cuadrado de la tabla ($\chi 2$ (α, N-1)) entonces hay concordancia:

$$\chi 2 \text{ real} < \chi 2 (\alpha, N-1)$$

$$\chi 2 \text{ real} < \chi 2(0{,}05, 4)$$

$$0.32 < 9{,}4877$$

Los resultados obtenidos corroboran el cumplimiento de la comparación y por tanto la existencia de concordancia entre los expertos.

**Reportes de los resultados**

El 89.7% de los expertos consideraron de Muy Adecuada la interfaz del módulo, donde la arquitectura de información permite localizar el contenido deseado con facilidad, logrando así un mejor acceso a la información.

La calidad de las funcionalidades fueron valoradas por el 67.5% de los expertos como Muy adecuada y por el resto de los expertos como Bastante Adecuada.

El consumo de la RAM, la SWAP y la CPU fue considerado por el 78.9% de los expertos que influye en el funcionamiento del servidor.

Un 100% de los expertos valoraron que el módulo contribuye al monitoreo en tiempo real de los clientes ligeros y favorece el control de los recursos de *hardware* en el servidor.

De las 40 posibles respuestas con la más alta calificación que podían otorgar los expertos, se recibieron 36, por lo que se concluye que hubo consenso en un 90% respecto a considerar como Muy adecuado el módulo para el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP. Este resultado permite concluir que el módulo favorece el control de los recursos de *hardware* en el servidor al finalizar los procesos que ejecuta el usuario, apagar los clientes ligeros y supervisar el consumo de la RAM, la SWAP y la CPU.

## 3.5 Valoración económica, novedad y aporte social

Los lineamientos 131, 152, 223, 226 y 228 de la política económica y social del Partido y la Revolución favorecen la integración de la ciencia, la tecnología, y la innovación en función del desarrollo social y económico del país. En consecuencia el vínculo universidad-empresa juega un papel fundamental, poniendo al servicio de las necesidades empresariales, a los profesionales y estudiantes de las universidades.

El módulo desarrollado en la presente investigación forma parte del convenio entre la UCI y la Empresa Industrial para la Informática, las Comunicaciones y la Electrónica (GEDEME) entidad que ensambla y comercializa clientes ligeros con la Distribución cubana GNU/LINUX Nova desarrollada en la universidad. El sistema le permitirá a un administrador con mínimos esfuerzos, administrar los clientes ligeros conociendo el porciento de recursos de *hardware* que están consumiendo en el servidor, la vista del escritorio remoto, así como finalizar los procesos que ejecuta el usuario. Además sirve de apoyo para aquellas empresas que no posean un capital muy amplio para sus inversiones y deseen informatizar algunos de sus procesos lo puedan hacer mediante la utilización de clientes ligeros, computadoras con un *hardware* generalmente barato.

## 3.6 Conclusiones parciales

En este capítulo se han abordado los elementos de la implementación del módulo de monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, así como las pruebas realizadas al mismo y los resultados obtenidos; arribando a las siguientes conclusiones:

- ✓ La elaboración de los diagramas de componentes, permitió una mejor comprensión de la estructura de los componentes del sistema implementado.

- ✓ El correcto uso de los estándares de codificación permitió que el código del sistema desarrollado fuera legible para lograr una fácil y mejor comprensión del mismo, la cual es de utilidad para el mantenimiento del sistema.
- ✓ La aplicación de las pruebas de unidad, integración, validación y rendimiento a la solución desarrollada permitió encontrar errores que afectaban el funcionamiento del sistema, lo que posibilitó corregirlos a tiempo para que el mismo cumpliese totalmente con los requisitos funcionales definidos en la etapa de análisis.
- ✓ La aplicación del método de criterio de expertos, específicamente el método Delphi, demostró la concordancia entre los expertos seleccionados. Se valida que el Módulo para el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, favorece el control de los recursos de *hardware* en el servidor.

## Conclusiones Generales

De manera general se puede concluir sobre la presente investigación:

- ✓ Las relaciones existentes entre los principales conceptos asociados al dominio de la presente investigación, permitieron una mayor comprensión de la propuesta de solución.
- ✓ El análisis de las diferentes herramientas para la administración de los clientes ligeros permitió determinar las características que constituyen la base para el diseño de las funcionalidades que se definen en la propuesta de solución.
- ✓ La elaboración de los artefactos propuestos por la metodología de desarrollo y la especificación de requisitos permitieron un mejor entendimiento del módulo que se propone desarrollar, así como las características del mismo.
- ✓ Como resultado de la implementación se obtuvo el Módulo para el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, que cumple con los 10 requerimientos funcionales identificados en la fase de ejecución.
- ✓ Las pruebas diseñadas y ejecutadas arrojaron como resultado que el sistema implementado responde a los requerimientos definidos por el cliente.
- ✓ La aplicación del método Delphi demostró la concordancia entre los expertos seleccionados y validó que el Módulo para el monitoreo en tiempo real de los clientes ligeros desde Nova-LTSP, favorece el control de los recursos de *hardware* en el servidor.

## Recomendaciones

Una vez concluida la investigación y el desarrollo de la propuesta de solución, el autor del presente trabajo recomienda:

- ✓ Incorporar al módulo funcionalidades para el monitoreo del escritorio remoto en tiempo real basado en el protocolo VNC.
- ✓ Incorporar al módulo un mecanismo de notificación de mensajes entre el servidor y el cliente.

**Referencias Bibliográficas**

1. PÉREZ, Renier. *Impacto de la Informatización en la Sociedad Cubana. Ciencia, tecnología y sociedad.* 2005.

2. GUEVARA, Yurisander. Las bases estratégicas de la informatización cubana (+ Video y PDF) - Informática - Suplementos - Juventud Rebelde - Diario de la juventud cubana. [online]. [Accessed 10 September 2016]. Available from: http://www.juventudrebelde.cu/suplementos/informatica/2015-02-18/las-bases-estrategicas-de-la-informatizacion-cubana/.

3. PÉREZ Villazón, Yoandy. *Metodología para la Migración a Software Libre de las Universidades del Ministerio de Educación Superior*. Tesis de diploma. La Habana, Cuba: Universidad de las Ciencias Informáticas, 2008.

4. Misión | Portal de la Universidad de las Ciencias Informáticas. [online]. [Accessed 11 October 2016]. Available from: http://www.uci.cu/?q=mision.

5. PÉREZ Villazón, Yoandy. *Estrategia para la migración a aplicaciones de código abierto.* Tesis de maestría. Universidad de las Ciencias Informáticas, 2015.

6. TOALOMBO Ortiz, Vicente Agustín. *Estudio comparativo de la tecnologías de clientes ligeros LTSP,TCos, Microsoft Terminal Server orientado a reutilización de PC' S. Caso práctico: Laboratorio de cómputo de la escuela Ruffo Diddonato.* Tesis de grado. Riobamba-Ecuador : Facultad de Infomática y Electrónica, Escuela Ingeniería en sistemad, 2012.

7. LÓPEZ, David. *Aplicación Práctica de Algoritmos Genéticos al Diseño de Redes.* [online]. [Accessed 11 October 2016]. Available from: https://www.iiisci.org/journal/CV$/risci/pdfs/P742867.pdf.

8. Virtualización clientes ligeros. [online]. 15 October 2016. [Accessed 15 October 2016]. Available from: http://www.slideshare.net/IsidroParga/virtualizacin-clientes-ligeros.

9. MOYA Moirán, Luis Miguel. *Tecnología Thin Client* [online]. 2010. [Accessed 15 October 2016]. Available from: https://lancore.sourceforge.net/doc/tecnologia_thin_client.pdf.

10. Arquitectura cliente servidor. [online]. [Accessed 15 October 2016]. Available from: http://www.slideshare.net/NoeGonzalezMendoza/arquitectura-cliente-servidor.

11. TCOS Project (thin client operating system). [online]. [Accessed 20 October 2016]. Available from: http://www.tcosproject.org.

12. Thin Client Server and Operating System. [online]. 20 October 2016. [Accessed 20 October 2016]. Available from: http://lancore.sourceforge.net/en/

13. Free OpenSource ThinClient Solution - openthinclient. [online]. [Accessed 21 October 2016]. Available from: http://openthinclient.org/en/open-thinclient/.

14. Linux Terminal Server Project - Welcome to LTSP.org. [online]. [Accessed 21 October 2016]. Available from: http://www.ltsp.org/.

15. SSH Communications Security - The Origin of SSH. [online]. [Accessed 24 October 2016]. Available from: https://www.ssh.com/.

16. BARZANALLA, Rafael. Apuntes. Ingeniería del software. Sistemas Informáticos. Nivel de madurez software. Informática Aplicada a la Gestión Pública. [online]. [Accessed 24 October 2016]. Available from: http://www.um.es/docencia/barzana/IAGP/Iagp2.html.

17. Universidad de las Ciencias Informáticas. Metodología de desarrollo para la Actividad productiva de la UCI. 2015.

18. Definición de Herramienta - Significado y definición de Herramienta. [online]. [Accessed 26 October 2016]. Available from: https://www.mastermagazine.info/termino/5234.php.

19. RUMBAUGH, James, JACOBSON, Ivar and BOOCH, Grady. *El lenguaje Unificado de Modelado. Manual de Referencia.* [no date].

20. LÓPEZ, Patricia and RUÍZ, Francisco. *Lenguaje Unificado de Modelado - UML*. 2011.

21. PRESSMAN, Roger. *Ingeniería de Software: Un Enfoque Práctico*. Quinta Edición. Nueva York,. Estados Unidos: McGraw-Hill, 2002.

22. Visual Paradigm. [online]. [Accessed 26 October 2016]. Available from: https://www.visual-paradigm.com/

23. GALINDO, R. *Guión Visual Paradigm for UML*. 2013.

24. Lenguaje_de_programación. [online]. [Accessed 26 October 2016]. Available from: http://dictionnaire.sensagent.leparisien.fr/Lenguaje_de_programaci%C3%B3n/es-es/.

25. ROSSUM, Guido. *El tutorial de Python.* [online]. 2009. [Accessed 26 October 2016]. Available from: http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf.

26. DAYLEY, Brad. *Python Phrasebook: Essential Code and Commands Recuperado* [online]. 2006. [Accessed 26 October 2016]. Available from: http://shodan.me/books/Programming/Python/Python%20Phrasebook%20%20Essential%20Code%20and%20Commands%20%282006%29.pdf.

27. KABIR, Mohammed J. *La biblia de Servidor Apache 2* [online]. 2002. [Accessed 26 October 2016]. Available from: https://yexia.files.wordpress.com/2010/09/mohammed-j-kabir-la-biblia-del-servidor-apache-21.pdf.

28. PyCharm. [online]. [Accessed 26 October 2016]. Available from: https://www.jetbrains.com/pycharm/.

29. MATOS García, Rosa María. *Diseño de almacenes de datos.*

30. Sobre PostgreSQL | www.postgresql.org.es. [online]. [Accessed 27 November 2016]. Available from: http://www.postgresql.org.es/sobre_postgresql.

31. GUTIÉRREZ, Javier. *Framework* [online]. [Accessed 27 November 2016]. Available from: www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf27-11-2016.

32. INFANTE MONTERO, Sergio. *Curso Django para perfeccionistas con deadlines* [online].2012. [Accessed 27 November 2016]. Available from: https://www.academia.edu/9510572/maestrosdelweb_curso_django

33. HOLOVATY, Adrián and KAPLAN-MOSS, Jacob. *The Definitive Guide to Django Web Development Done Right*. 2009.

34. OTTO, Mark and THORNTON, Jacob. *Bootstrap 3, el manual oficial - LibrosWeb.es* [online]. [Accessed 27 November 2016]. Available from: http://librosweb.es/libro/bootstrap_3/

35. Manual de jQuery. [online]. [Accessed 27 November 2016]. Available from: http://www.desarrolloweb.com/manuales/manual-jquery.html

36. LARMAN, Craig. *UML y patrones. Segunda Edición* [online]. 2003. [Accessed 28 November 2016] . Available from: http://is.ls.fi.upm.es/docencia/is2/documentacion/ModeloDominio.pdf

37. JACOBSON, Ivar, BOOCH, Grady and RUMBAUGH, James. *El Proceso Unificado de Desarrollo de Software*.

38. SOMMERVILLE, Ian. *Software Engineering*. 2007.

39. CANÓS, José, LETELIER, Patricio and PANADÉS, Ma Carmen. *Metodologías Ágiles en el Desarrollo de Software*. 2006.

40. NÚÑEZ, José. *Usabilidad en metodologías Ágiles*. Tesis de maestría. Facultad de Informática, España : Universidad Politécnica de Madrid.

41. PRESSMAN, Roger. *Software Engineering: A practitioner´s Approach.* Seventh Edition. New York. Estados Unidos: McGraw-Hill, 2010.

42. LARMAN, Craig. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall, 1999. ISBN 970-17-0261-1.

43. Sparx Systems - Tutorial UML 2 - Diagrama de Despliegue. [online]. 2013. [Accessed 28 February 2017]. Available from: http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

44. *Guía de estilo para el código Python-PEP 8 en Español* [online]. 2013. [Accessed 28 February 2017]. Available from: www.recursospython.com/pep8es.pdf

45. PRESSMAN, Roger. *Ingeniería de Software: Un Enfoque Práctico.* Sexta Edición. Nueva York. Estados Unidos: McGraw-Hill / Interamericana de México, 2005.

46. SOMMERVILLE, Ian. *Ingeniería del software*. 7ma Edición. 2005. ISBN ISBN 84-7829-074-5.

47. Apache JMeter. [online]. 2015. [Accessed 30 March 2017]. Available from: http://jmeter.apache.org/.

48. SÁNCHEZ Ortiz, Susana. *Estrategia de soporte técnico para el proceso de migración a código abierto en los Organismos de la Administración Central del Estado.* Tesis de maestría. La Habana, Cuba: Universidad de las Ciencias Informáticas, 2015.

49. GARCÍA Valdés, Margarita and SUÁREZ Marín, Mario. *El método Delphi para la consulta a expertos en la investigación científica.Universidad de Ciencias Médicas de la Habana.* 2012.

50. HERNÁNDEZ Sampier, Roberto and FERÁNDEZ Collado, Carlos. *Metodología de la investigación*. McGrawHill, 2010.