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

Sistema para la comunicación con dispositivos desde aplicaciones web en GNU/Linux

Autor: Joseph Michael Cordero Korolev

Tutor(es): MSc. Juana Elena Acosta García

Ing. Ivan Campos Cesar

Ing. Kilmer Hernández Avila

La Habana, junio 2016

"…Dime y lo olvido, enséñame y lo recuerdo, involúcrame y lo aprendo…"

Benjamín Franklin

Declaración de autoría

Declaro que soy el único autor del presente trabajo titulado:
Sistema para la comunicación con dispositivos desde aplicaciones *web* en *GNU/Linux* v1.0. Y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Autor: Joseph Michael Cordero Korolev.

_____

Tutor: MSc. Juana Elena Acosta García

_____

Tutor: Ing. Iván Campos Cesar.

_____

Tutor: Kilmer Hernández Avila.

_____

Resumen

La interacción con dispositivos conectados a estaciones de trabajo posibilita entre otras cosas, la captura de imágenes a través de una cámara *web* o simplemente la impresión de documentos de cualquier tipo de categoría. Sin embargo, en el caso de las aplicaciones *web*, dependen del uso de las tecnologías *web* del lado del cliente para la interacción con dispositivos, que se limitan a las bondades del navegador utilizado y no cuentan con mecanismos de acceso a los recursos locales.

Como solución a este problema se decide realizar un sistema para la comunicación con dispositivos desde aplicaciones *web* en *GNU/Linux*[1] que posibilite la interacción entre dispositivos de *hardware* [2] y aplicaciones *web* del lado del cliente, de manera independiente del navegador, lo cual garantice la seguridad y el control centralizado de estos dispositivos para cubrir las necesidades que afronta el país debido al proceso de migración por el cual, están transitando las principales organizaciones e instituciones.

Para ello se lleva a cabo un estudio y valoración de los sistemas existentes a nivel nacional e internacional, a fin de encontrar opciones y elementos a tomar en cuenta para obtener la solución deseada. Para el modelado y desarrollo del sistema se utilizaron las herramientas *Visual Paradigm* y *Netbeans* y los lenguajes UML, Java 8, HTML 5 y *Java Script* respectivamente. Este desarrollo fue guiado por la metodología XP y las fases de planificación, diseño e implementación. Por medio de las pruebas de aceptación y camino básico se garantizó el correcto funcionamiento del sistema.

Palabras clave: aplicaciones *web*, control centralizado, dispositivos e interacción.

[1] *GNU/Linux:* Sistema operativo libre homólogo a Unix que usualmente utiliza herramientas de Sistema GNU.

[2] *hardware:* Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.

Índice de contenido

Índice de tablas

Índice de Ilustraciones

Introducción

Durante el desarrollo de la informática, la historia ha recogido diferentes etapas. El uso de las computadoras como ente que almacena, recupera y procesa datos que se convierten en información valiosa da paso a la existencia de la comunicación asistida por computadoras. En la actualidad, esta nueva forma de gestión tiene gran impacto, puesto que para un buen desempeño de una administración es sumamente importante el uso de toda la información que se pueda tener para el manejo y la toma de decisiones. Sirviendo además de apoyo para llevar a cabo los diseños organizacionales, los que van desde pequeños grupos hasta los de las grandes organizaciones e instituciones que rigen a nivel mundial.

Sin embargo, la comunicación usuario-usuario no es el único método empleado por estas organizaciones e instituciones para lograr un ascendente proceso de desarrollo; la comunicación usuario-dispositivo, dispositivo-dispositivo, aplicación-dispositivo, por citar algunos ejemplos, y cómo lograrlo son las claves principales para evolucionar en cualquiera de las etapas de desarrollo.

La comunicación con los dispositivos de *hardware* ha sido uno de los pilares fundamentales para el desarrollo en las grandes organizaciones e instituciones, en las cuales prima el uso de la informática como base para alcanzar dicho desarrollo. Estas organizaciones e instituciones han permanecido en una lucha constante para lograr un control equilibrado sobre sus Tecnologías Informáticas (TI)[3]. Muchas de éstas utilizan dispositivos de *hardware* que le permiten en muchos casos el control de acceso al personal, realizar capturas de imágenes, la obtención de la información de las estaciones de trabajo o simplemente la impresión de documentos de todo tipo. Una de las vías que han encontrado estas organizaciones e instituciones para lograr el control centralizado de los dispositivos de *hardware* -Es decir, desde una estación de trabajo controlar los dispositivos conectados a una subred- es la utilización de aplicaciones *web* que permitan la gestión de los mismos.

Sin embargo, en el intento de sustituir las aplicaciones de escritorio; las aplicaciones *web* se han encontrado con una barrera que les impide integrar dispositivos de *hardware* con las aplicaciones en los clientes. Las soluciones *web* se ven restringidas a utilizar tecnologías *web* del lado del cliente para la interacción con los dispositivos de *hardware* conectados a las estaciones de trabajo. Por su parte, estas tecnologías están limitadas

3 Tecnologías Informáticas (TI): Hace referencia a las herramientas informáticas presentes en las organizaciones e instituciones.

a las bondades del navegador *web* que se esté utilizando, y por razones de seguridad estos no cuentan con un mecanismo de acceso a los recursos locales de forma directa.

Las alternativas con que cuentan los navegadores para la interacción con recursos locales son escasas y no se encuentran estandarizadas. Las más formales sólo permitirían el control administrativo de los dispositivos de *hardware* mientras exista una instancia del navegador funcionando, ya que su ejecución dependería totalmente del navegador *web*, por lo que no podría ser posible el control total de los dispositivos en cada momento.

De ahí que las grandes organizaciones o grupos de desarrollo de *software* han encaminado su trabajo en implementar, para luego lanzar al mercado aplicaciones que permitan a las aplicaciones *web* interactuar con los dispositivos de *hardware* conectados a las estaciones de trabajo, así como obtener el control centralizado de los mismos independientemente del navegador *web* que se esté utilizando.

Con la aparición del código abierto, gran parte del mundo se ha proyectado hacia la soberanía tecnológica. Desde simples aplicaciones hasta Sistemas Operativos (SO) como *GNU/Linux* han ido tomando posición en las organizaciones e instituciones, sin embargo, en el campo de la comunicación y control desde aplicaciones *web* de los dispositivos de *hardware* conectados a las estaciones de trabajo, no han sido homologadas la mayoría de las aplicaciones existentes en el SO *Windows*[4]. Muchas de las ya desarrolladas permiten la comunicación con algunos de los dispositivos y otras, muy pocas, posibilitan el control de un tipo de dispositivo específico.

Actualmente, Cuba está inmersa en un proceso de informatización de la sociedad, en el cual, la Universidad de las Ciencias Informáticas (UCI) es vanguardia. La migración de algunas de las principales organizaciones e instituciones del país ha sido uno de los logros de dicho proceso. Para lograr un control centralizado de los dispositivos de *hardware* conectados a las estaciones de trabajo desde aplicaciones *web* el Centro de Identificación y Seguridad Digital (CISED) de la UCI desarrolló la aplicación *Device Grid Manager* (DGM) y aunque esta aplicación solventa los problemas antes mencionados es totalmente privativa por lo que no se garantiza su uso en estas organizaciones.

La limitante mencionada hace necesario el desarrollo de una solución con tecnologías libres, que permita a las aplicaciones *web* el control en tiempo real y la administración centralizada de los dispositivos de *hardware* conectados a las estaciones de trabajo en

[4] *Windows: es un grupo de sistemas operativos diseñados y comercializados por la empresa Microsoft.*

Introducción

el SO *GNU/Linux* debido a la necesidad de incrementar la seguridad de los sistemas informáticos y específicamente en las aplicaciones *web*.

Teniendo en cuenta que, muchas aplicaciones *web* en la actualidad que trabajan con dispositivos de *hardware* y destinadas a la administración de controles de acceso, además de la administración y gestión de identidades y fichas policiales presentan un control de acceso tradicional como RBAC o a través de permisos que puedan ser configurados en un centro de datos sobre las direcciones IP y físicas de la máquina o, una comunicación directa de la solución *web* a los dispositivos de *hardware* instalados en las estaciones de trabajo cliente, lo cual puede presentar posibles problemas de integridad en la información y de seguridad en sentido general.

Un ejemplo claro de lo mencionado anteriormente puede ser el acceso a la aplicación con una máquina personal de un usuario de la aplicación que se conecte a la red y ponga una dirección IP con acceso y haga un duplicado de la MAC, además de la posible utilización de cámaras que no estén contempladas por el sistema y emita imágenes o videos sin la calidad requerida.

Atendiendo a la situación planteada, se identifica como **problema de investigación:** ¿Cómo posibilitar la comunicación de aplicaciones *web* con dispositivos de *hardware* y su control centralizado en *GNU/Linux*?

Una vez planteado el problema de investigación se define como **objeto de estudio** los procesos de comunicación de dispositivos de *hardware* con aplicaciones *web*; enmarcando el **campo de acción** de la investigación en los procesos de interacción de aplicaciones *web* con dispositivos de *hardware* en *GNU/Linux*.

El **objetivo general** está enfocado en desarrollar un sistema que posibilite la interacción de aplicaciones *web* con dispositivos de *hardware* y su control centralizado en *GNU/Linux.*

Para dar cumplimiento al objetivo planteado se proponen las siguientes **tareas:**

1) Análisis de los principales conceptos asociados a los procesos de comunicación y control de dispositivos para obtener una base teórica necesaria para el desarrollo de la solución.
2) Análisis de las soluciones existentes que permiten la comunicación con los dispositivos, para determinar fortalezas e insuficiencias en las mismas y valorar la vialidad de su uso.
3) Especificación de los requisitos del sistema para el manejo de dispositivos, con el objetivo de determinar las funcionalidades a implementar.

4) Realización de un estudio que permita definir cuáles son las herramientas informáticas y metodologías a usar para el desarrollo del sistema.
5) Diseño de la solución del sistema para el manejo de dispositivos.
6) Diseño e implementación de las interfaces de la aplicación.
7) Implementación de un servicio que permita el manejo de la *webcam* en las estaciones de trabajo.
8) Implementación de un servicio que permita el manejo de la impresora en las estaciones de trabajo.
9) Implementación de un servicio que permita la obtención de la información básica de las estaciones de trabajo.
10) Desarrollo de un servicio que garantice mecanismos de autenticación-autorización entre los servicios.
11) Aplicación de las pruebas de *software*[5] a la solución.

Para llevar a cabo las tareas de la investigación se emplearán los siguientes **métodos científicos:**

- **Histórico-Lógico:** Al comienzo de la investigación se desarrollará un estudio del estado del arte y se realizará el análisis de las ventajas y desventajas de las herramientas utilizadas actualmente que permitan a las aplicaciones *web* el control centralizado de los dispositivos de *hardware* conectados a las estaciones de trabajo.
- **Análisis-Sintético:** Se utilizará para el procesamiento de la bibliografía necesaria para llevar a cabo la investigación.
- **Modelación:** Permitirá la creación de modelos (propuestas, alternativas y estrategias) que visualizan una reproducción simplificada de la realidad.

Al término de la investigación se contará con un sistema que servirá de intermediario para lograr la comunicación entra aplicaciones *web* y los dispositivos de *hardware* conectados a las estaciones de trabajo donde se encuentre instalado el servicio, así como la obtención de la información básica de las mismas.

El presente trabajo de diploma consta de 3 capítulos, los cuales se describen brevemente a continuación:

El **Capítulo I** titulado **Fundamentación teórica**, en el cual se describen todos los elementos de la teoría que sostienen el problema de investigación y los objetivos de la misma. Resumiendo, el estado del arte, se analizan y fundamentan las tecnologías, lenguajes de programación, herramientas y metodologías para validar y guiar el proceso

[5] *software:* Se refiere al equipamiento lógico o soporte lógico de una computadora digital.

de desarrollo del *software* requerido.

El **Capítulo II** titulado **Características de sistema**, se describen teniendo en cuenta la especificación de los requerimientos no funcionales, historias de usuarios, diagramas de clases, del análisis y del diseño, así como la descripción de la arquitectura y patrones de diseños utilizados.

El **Capítulo III** titulado **Implementación y validación de la solución propuesta**, en el mismo se expone el diagrama de componentes, además se realiza la validación de la solución propuesta, mediante la realización de pruebas y métricas de calidad, además de la comprobación del cumplimiento del objetivo general de la investigación.

Capítulo I: Fundamentación Teórica

1.1 Introducción

El presente capítulo contiene la fundamentación teórica de la investigación realizada y que da respaldo a la solución propuesta. Se muestran las definiciones y conceptos tales como: sistema, comunicación, dispositivos, servicios *web*, *daemons* y *GNU/Linux*. Se hace además un estudio de sistemas y aplicaciones ya existentes especializadas en la comunicación, control y administración de dispositivos. Se analiza el ambiente de desarrollo propuesto, caracterizando las herramientas y tecnologías que se definieron para ser utilizadas en la confección de la solución propuesta.

1.2 Conceptos generales

1.2.1 Sistema

El enfoque de sistema lo sustenta el principio filosófico en el que se expresa que todo fenómeno de la realidad objetiva debe ser considerado desde posiciones de las leyes del todo sistémico y la interacción de las partes que lo forman. Esta noción conlleva a considerar que se trata de un conjunto de cuerpos (elementos del sistema) entre los cuales se dan determinadas relaciones que le otorgan un carácter de totalidad. Esta reflexión del autor es resultado de sus estudios realizados en la disciplina Ciencias Sociales.

Un sistema informático es un conjunto de partes que funcionan relacionándose entre sí con un objetivo preciso. Sus partes son: *hardware*, *software* y las personas que lo usan (1).

El *hardware* consta de las computadoras, discos, impresoras, cámaras *web*, enrutadores, y en general de todos los elementos que una computadora requiere para funcionar.

El *software* está formado por el SO, los compiladores, los programas de aplicación, los sistemas administradores de base de datos y los programas de oficina, entre otros.

El elemento humano está formado por los operadores y el equipo de mantenimiento del sistema informático.

1.2.2 Comunicación

De manera general es el proceso de transmisión y recepción de ideas, información y mensajes. El acto de comunicar es un proceso complejo en el que dos o más personas

se relacionan y, a través de un intercambio de mensajes con códigos similares, tratan de comprenderse e influirse de forma que sus objetivos sean aceptados en la forma prevista, utilizando un canal que actúa de soporte en la transmisión de la información. Es más, un hecho sociocultural que un proceso mecánico (2).

Es criterio del investigador que la comunicación entre dispositivos no es más que el intercambio de datos entre los mismos que se genera a través de códigos embebidos en las aplicaciones o servicios conectados entre sí, ya sea en el mismo SO o a través de la red.

1.2.3 Dispositivos

Es criterio del investigador que los dispositivos no son más que aparatos o periféricos independientes conectados a la unidad central de procesamiento de una computadora. O sea, son las unidades a través de las cuales la computadora se comunica con el mundo exterior.

Clasificación

Los dispositivos pueden clasificarse en cinco categorías principales (3):

- **Dispositivos de entrada:** captan, y digitalizan los datos de ser necesario, introducidos por el usuario o por otro dispositivo y los envían al ordenador para ser procesados.
- **Dispositivos de salida:** son dispositivos que muestran o proyectan información hacia el exterior del ordenador. La mayoría son para informar, alertar, comunicar, proyectar o dar al usuario cierta información, de la misma forma se encargan de convertir los impulsos eléctricos en información legible para el usuario. Sin embargo, no todos de este tipo de periféricos es información para el usuario.
- **Dispositivos de entrada/salida (E/S):** sirven básicamente para la comunicación de la computadora con el medio externo. Los dispositivos de entrada/salida son los que utiliza el ordenador tanto para mandar como para recibir información. Su función es la de almacenar o guardar de forma permanente o virtual todo lo que se realice con el ordenador para que pueda ser utilizado por los usuarios u otros sistemas.
- **Dispositivos de almacenamiento:** son los dispositivos que almacenan datos e información por bastante tiempo. La memoria de acceso aleatorio no puede ser considerada un periférico de almacenamiento, ya que su memoria es volátil y temporal.

- **Dispositivos de comunicación:** son los periféricos que se encargan de comunicarse con otras máquinas o computadoras, ya sea para trabajar en conjunto, o para enviar y recibir información.

1.2.4 Servicios *web*

Conjunto de aplicaciones o de tecnologías con capacidad para operar de manera conjunta o individual en la *web*. Estas intercambian datos entre sí con el objetivo de ofrecer servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios los solicitan llamando a estos procedimientos a través de la *web*. A su vez proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar (3).

Estándares empleados

***Extensible Markup Language* (XML):** es un lenguaje de marcas desarrollado por el *World Wide Web Consortium (W3C)*[6] utilizado para almacenar datos en forma legible. Permite definir la gramática de lenguajes específicos para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información (4).

***Simple Object Access Protocol* (SOAP):** es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Básicamente es un paradigma de mensajería de una dirección sin estado, que puede ser utilizado para formar protocolos más complejos y completos según las necesidades de las aplicaciones que lo implementan (5).

***Web Services Description Language* (WSDL):** describe la interfaz pública a los servicios *web*. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje (6).

[6] *World Wide Web Consortium (W3C):* Es una comunidad internacional donde la organización miembro, un equipo de trabajo a tiempo completo, y el público, trabajan de conjunto para desarrollar estándares *web*.

***Representational State Transfer* (REST):** es un conjunto de principios, o maneras de hacer las cosas, que define la interacción entre distintos componentes, es decir, las reglas que dichos componentes tienen que seguir. El protocolo más usado que cumple esta definición, es el protocolo HTTP. Además, proporciona una API que utiliza cada uno de sus métodos (*GET*, *POST*, *PUT*, *DELETE*, etc.) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio *web* y el cliente (7).

1.2.5 *Daemons*

Daemon o demonio es un programa o proceso que constantemente está disponible y permanece inactivo hasta que es invocado para cumplir una tarea. Un *daemon* luego puede distribuir la tarea recibida a otros programas o procesos (8).

Este término está más asociado a servidores con el SO de *UNIX*[7], y normalmente el usuario nunca interactúa directamente con éstos. Algunas actividades que pueden activar un *daemon* pueden ser la invocación de un servicio *web* en un canal de comunicación, cuando se recibe un archivo a un directorio determinado, cuando se recibe un correo electrónico, etc.

Daemon es una de las tres clasificaciones de cómo puede ser catalogado un proceso. Los otros dos estilos de procesos son conocido como *Batch* (procesos que se acumulan en un periodo de tiempo para ejecutarse), o Interactivo (donde el usuario interactúa con una línea de comando). En *Windows*, estos procesos se conocen como servicios.

1.2.6 *GNU/Linux*

GNU/Linux es, a simple vista, un SO. Es una implementación de libre distribución *UNIX* para computadoras personales (PC), servidores, y estaciones de trabajo. Fue desarrollado para el i386 y ahora soporta los procesadores i486, *Pentium*[8], *Pentium Pro y Pentium II*, etc., así como los clones *AMD y Cyrix*[9].

Como SO, GNU/Linux es muy eficiente y tiene un excelente diseño. Es multitarea, multiusuario, multiplataforma y multiprocesador; en las plataformas Intel corre en modo protegido; protege la memoria para que un programa no pueda hacer caer al resto del sistema; carga sólo las partes de un programa que se usan; comparte la memoria entre programas aumentando la velocidad y disminuyendo el uso de memoria; usa un sistema de memoria virtual por páginas; utiliza toda la memoria libre para cache; permite usar

[7] Unix (registrado oficialmente como UNIX®): Es un SO portable, multitarea y multiusuario.
[8] *Pentium: Intel Pentium* es una gama de microprocesadores de quinta generación con arquitectura x86 producidos por Intel Corporation.
[9] *Cyrix:* Se refiere a una gama de microprocesadores fabricados por una empresa de igual nombre.

bibliotecas enlazadas tanto estática como dinámicamente; se distribuye con código fuente; usa hasta 64 consolas virtuales; tiene un sistema de archivos avanzado pero puede usar los de los otros sistemas; y soporta redes tanto en TCP/IP como en otros protocolos (9).

1.3 Análisis de los sistemas existentes

En la actualidad existen sistemas informáticos que se encargan del monitoreo de dispositivos de *hardware*, posibilitando de esta forma el trabajo de las organizaciones ya que pueden manejar variedades de dispositivos, entre otras funciones. A continuación, se realiza un estudio de los mismos tanto internacionales como nacionales, en busca de un sistema que cubra la mayor parte de las necesidades del problema a resolver.

1.3.1 Sistemas Internacionales

nddPrint MPS

Es un sistema de administración de la producción especializado en la gestión y operación de parques de impresión. El sistema trabaja en una plataforma en la nube, factor que permite que las operaciones sean gestionadas de manera centralizada, para parques de impresión de alta o baja distribución. Además, dispone de recursos para gestionar todas las operaciones de las organizaciones que mantengan parques de *outsourcing* de impresión. Con la solución *nddPrint MPS* es posible realizar un control integral, entre estos controles es posible mencionar (10):

- Gestión y administración de los *stocks* de la producción.
- Gestión y administración de la producción de páginas impresas.
- Gestión y administración de eventos de las impresoras.
- Gestión y administración de la flota de impresoras.
- Acompañamiento en línea del contrato.

La gestión de estos recursos es enfocada en la eficiencia, control y análisis de rentabilidad del negocio.

Ventajas

- Monitorización y pro-actividad en la detección de fallas de las impresoras y el del entorno.
- Permite el almacenamiento en la nube.
- Posibilita programar los reportes respectando la jerarquía de centro de coste, alimentada de forma automática, además de permitir acceso a los reportes solamente por su respectivo usuario.
- Es multiplataforma y permite el control en tiempo real de los dispositivos a través

de una interfaz *web*.

Al disponer de una interfaz *web* es posible realizar todas las configuraciones, así como la utilización de todas las herramientas de gestión sin inversiones pesadas en servidores, sistemas operacionales y banco de datos. Dicho panel ofrece la generación en línea de informes con variedades de filtros avanzados y, además de eso, trae comodidad, pues todos los informes tienen opción de poner fecha para generación y envío automático, pudiendo ser un único día, semanalmente o mensualmente (10).

CZ Print Job Tracker 9.0

Es un *software* de gestión de impresión fácil de usar, permite gestionar, limitar y contar las impresiones, así como supervisar y llevar un registro de todas las actividades de impresión, analizar y monitorear los costos, facturar los trabajos indicando los códigos del cliente o del proyecto en la estación de trabajo, confirmar y autentificar los trabajos en la estación antes de su impresión efectiva, eliminar el desperdicio de papel y reducir el tiempo de mantenimiento de la impresora. Además, recoge la información detallada de cada trabajo de impresión, incluyendo el nombre del usuario que lo envió, el nombre del documento, el número de páginas, el coste total, la hora y la fecha de envío, la impresora de destino, el nombre de la computadora de origen, dúplex, color, tamaño del papel y tamaño del trabajo (11).

Posee una amplia variedad de características entre las que destacan:

- Permite tener un control en tiempo real de los dispositivos.
- Fácil de instalar y usar ya que no requiere cambio alguno en la configuración de la red, la impresora o estación de trabajo.
- Permite la supervisión de las impresoras, o sea, identifica y limita las impresiones en los siguientes tipos de impresoras:
  - Impresoras en servidores de impresión de *Windows* / *Unix* / *Linux*.
  - Impresoras conectadas localmente a las estaciones de trabajo.
  - Impresoras conectadas a la red directamente a través de la dirección IP sin usar el servidor de impresión.
  - Impresoras conectadas con el dispositivo de la impresión.
- Facturación por cliente o proyecto**:** Los usuarios pueden facturar los trabajos indicando sus códigos de cliente/ de proyecto/ compartidos antes de que los trabajos se impriman en sus computadoras cliente *Windows, Mac OSX / Classic*[10], o *Linux*.

[10] Mac OSX/Classic: Es el nombre del SO creado por *Apple* para su línea de computadoras *Macintosh*.

Capítulo I

OCS Inventory

Open Computer and Software Inventory Next Generation (*OCS*) es un sistema para mantener el inventario de máquinas de forma fácil y automatizada. Es un programa sencillo pero muy útil. Permite tener un inventario centralizado de *software* y *hardware*; además de la disposición de forma rápida y completa de los datos de los equipos y su relación con los usuarios. *OCS Inventory* está basado en *software* libre y publicado bajo la licencia *GNU General Public License*[11], versión 2.0 (12).

El sistema *OCS Inventory* permite tener una vista centralizada de los servidores y computadoras de escritorio. También facilita las tareas de mantenimiento y renovación de *hardware*, la detección de *software* no autorizado y la prevención de ataques de seguridad brindando detalles de las versiones de los programas instalados en cada computadora.

A continuación, se muestran algunas de las características que posee el sistema (12):

- Permite tener un control en tiempo real de los dispositivos.
- Permite tener una vista centralizada de los servidores y computadoras de escritorio.
- Facilita la prevención de ataques de seguridad brindando detalles de las versiones de los programas instalados en cada estación de trabajo.
- El instalador del agente se encuentra disponible para *Windows y Linux*.
- Permite la interacción con los dispositivos a través de una interfaz *web*.

Pese a poseer una amplia variedad de características también posee algunas desventajas:

- No permite el control sobre sistemas *iOS*[12].
- Los servicios son implementados en *Windows Mobile* (en proceso).
- El Servidor requiere una pre-instalación de los servicios que necesita, antes de instalar el OCS Inventory.

1.3.2 Sistemas Nacionales

***Device Grid Manager* (DGM)**

Este sistema se encuentra corriendo sobre *software* privativo; por lo que es de primordial importancia su migración a tecnologías libres. Esta aplicación permite el control y comunicación de dispositivos de *hardware* en aplicaciones *web*. Está compuesto por

[11] *GNU General Public License:* Es la licencia más ampliamente usada en el mundo del *software* y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el *software*.

[12] iOS: *iPhone* OS como antes era denominado, es un SO desarrollado por *Apple*, inicialmente solo para el teléfono inteligente de la compañía (el *iPhone*), luego fue extendido a otros dispositivos como el *iPod Touch* y el *iPad*.

tres subsistemas fundamentales: Servicio DGM, el *framework*[13] *JavaScript* (DGMJS) y el control centralizado de dispositivos (13).

Funciones del sistema

Servicio de DGM: Es el encargado de garantizar el manejo de los servicios asociados a los dispositivos conectados a las estaciones de trabajo. El mismo contiene los controladores de la interacción directa con los dispositivos de *hardware* y gestiona las peticiones que provienen del navegador *web*. El Servicio de DGM posee tres partes fundamentales, el servidor proxy, el motor de manejo para los servicios asociados a los dispositivos y el grupo de clientes y controladores encargados de interactuar directamente con un dispositivo de *hardware*. Para el funcionamiento de esta versión del servicio local y llevar a cabo su instalación es necesario tener en cuenta los siguientes requisitos:

- *Microsoft Windows XP SP3* o superior.
- *Microsoft .Net Framework 4.0.*
- Controladores de los dispositivos a utilizar.

***framework DGMJS*:** Es un componente que cuenta con un grupo de clases que permiten al desarrollador la interacción con servicios publicados en la estación de trabajo donde se esté ejecutando el Servicio local para el manejo de dispositivos. De esta manera se establece una comunicación entre la aplicación web externa y el servicio local, posibilitando así interactuar con los dispositivos. El desarrollo de este *framework* es similar a la implementación de AJAX, una tecnología bien conocida y estandarizada, la cual es de conocimiento de un gran porciento de programadores web, por lo que el desarrollador no necesita asimilar un alto grado de conocimiento haciendo la curva de aprendizaje bien baja.

Control Centralizado: A través de una interfaz visual en forma de aplicación *web* permite al usuario gestionar los dispositivos de la red, así como ver en tiempo real el estado de estos. Se caracteriza por su fácil uso permitiéndose usar en prácticamente cualquier escenario.

1.3.3 Valoración de los sistemas analizados

Luego de realizado el análisis de los sistemas existentes en relación con el tema de investigación, se muestra a continuación en la tabla valorativa los criterios que a

[13] *framework* o infraestructura digital: Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de *software,* que puede servir de base para la organización y desarrollo de *software*.

consideración del investigador son necesarios para dar solución a la problemática planteada.

Sistemas	CTR[14]	ACD[15]	UAW[16]	CDM[17]	SOS[18]
nddPrint MPS	Sí	Sí	Sí	Impresora	*Windows/Linux*
CZ Print Job Tracker 9.0	Sí	Sí	No	Impresora	*Windows/Linux*
OSC Inventory	Sí	Sí	Sí	Información PC	*Windows/Linux*
DGM	Sí	Sí	Sí	Impresora Cámara *web* Información PC Firma digital Pasaporte Huellas Escáner	*Windows*
Sistema para el Control Centralizado de Dispositivos (SCCD)	Sí	Sí	Sí	Impresora Cámara *web* Información PC	*Linux*

Tabla 1: Sistemas analizados y sus características. Fuente: Elaboración Propia.

Pese a que los sistemas internacionales analizados son multiplataforma y que a excepción de *CZ Print Job Tracker 9.0*, permiten la interacción con los dispositivos a través de una interfaz *web*, estos sistemas son idóneos solamente para el control y gestión de las impresoras, a su vez el *OCS Inventory* solamente es utilizado para obtener la información de los componentes de *hardware* y *software* de un ordenador. En el caso de los sistemas nacionales, el DGM cumple con todas las características, pero el mismo está desarrollado solamente para su ejecución en plataformas privativas. Por lo que se llega a la conclusión de que estos sistemas no satisfacen las necesidades del problema de investigación y esto trae consigo que ninguna de las herramientas estudiadas puede ser adaptada al SCCD, aunque sí sirvieron de apoyo en la concepción de algunas interfaces y funcionalidades que por la complejidad del negocio no quedaban del todo claras en las bibliografías consultadas para el desarrollo del componente.

Por lo visto anteriormente cada uno de los sistemas presentados posee características

[14] CTR: Control en tiempo real de los dispositivos.
[15] ACD: Administración centralizada de los dispositivos.
[16] UAW: Uso en aplicaciones *web.*
[17] CDM: Cantidad de dispositivos que maneja.
[18] SOS: Sistema operativo que soporta.

diferentes, sirviendo su análisis como base para un mejor entendimiento de los rasgos que el nuevo sistema deberá presentar.

El estudio previamente realizado demuestra la necesidad de desarrollar una aplicación similar al sistema DGM en su versión 2.9.4, ya que de los sistemas estudiados es el que más dispositivos de *hardware* maneja, entre los que se incluyen: la impresora, la cámara *web* y la obtención de la información básica de las estaciones de trabajo. En este caso, la nueva entrega debe incluir los tres servicios mencionados anteriormente en aras de hacer del mismo un sistema integral para el manejo y control de dispositivos orientado a la arquitectura *web*, que provea herramientas para la administración de los dispositivos y su interacción desde aplicaciones *web*. Utilizando las experiencias de las aplicaciones estudiadas se pretende realizar un nuevo sistema, que permita la interacción de dispositivos en sistemas *GNU/Linux*, la continua actualización del sistema, la configuración y control centralizado de los dispositivos, la obtención de la información básica del *hardware* y el *software* de las estaciones de trabajo, logrando reducir, con el desarrollo del mismo, el costo al país.

1.4 Metodología

En la ingeniería de *software* la metodología de desarrollo de *software* no es más que el uso de un marco de trabajo para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Es un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que sirva de apoyo a los desarrolladores a la hora de diseñar un nuevo *software* (14).

El uso de estas metodologías, sirve de ayuda a los desarrolladores de *software* durante todo el ciclo de vida del mismo, por lo que para lograr una apropiada selección se deben tener en cuenta las características del *software* a obtener, el entorno para el que se desarrolla y el equipo de trabajo.

1.4.1 Metodología *Extreme Programming* (XP)

La **programación extrema** o *eXtreme Programming* es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck[19], autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace Change* (1999). Es el más destacado de los procesos ágiles de desarrollo de *software*. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural,

[19] Kent Beck: Es ingeniero de *software* estadounidense, uno de los creadores de las metodologías de desarrollo de *software* de programación extrema (*eXtreme Programming* o XP) y el desarrollo guiado por pruebas (*Test-Driven Development* o TDD), también llamados metodología ágil.

inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos (15).

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del *software*.

Valores

Los valores originales de la programación extrema son: simplicidad, comunicación, retroalimentación (*feedback*) y coraje. Un quinto valor, respeto, fue añadido en la segunda edición de Extreme Programming Explained. Los cinco valores se detallan a continuación (16).

Simplicidad: La simplicidad es la base de la programación extrema, se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores, hace que la complejidad aumente exponencialmente (16). Para mantener la simplicidad es necesaria la refactorización del código, ésta es la manera de mantener el código simple a medida que crece. También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso sin que el código esté autodocumentado. Para ello, se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo gracias a las herramientas de autocompletado y refactorización que existen actualmente. Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

Comunicación: La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad.

Capítulo I

Los programadores se comunican constantemente gracias a la programación por parejas. La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas (16).

Retroalimentación (*feedback*): Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real. Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

Coraje y valentía: Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementaran más fácilmente. Otro ejemplo de valentía es saber cuándo desechar un código: valentía para quitar código fuente obsoleto, sin importar cuanto esfuerzo y tiempo se invirtió en crear ese código. Además, valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, sólo si es persistente (16).

Respeto: El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o eficiente para la solución a través de la refactorización del código. Los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, una mejor autoestima en el equipo eleva su ritmo de reducción.

Capítulo I

Características fundamentales

Las características fundamentales de la metodología XP son:

- **Desarrollo iterativo e incremental:** En un desarrollo iterativo e incremental el proyecto se planifica en diversos bloques temporales llamados iteraciones. Las iteraciones se pueden entender como mini-proyectos: en todas las iteraciones se repite un proceso de trabajo similar (de ahí el nombre "iterativo") para proporcionar un resultado completo sobre producto final, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental. Para ello, cada requisito se debe completar en una única iteración: el equipo debe realizar todas las tareas necesarias para completarlo (incluyendo pruebas y documentación) y que esté preparado para ser entregado al cliente con el mínimo esfuerzo necesario (16).
- **Pruebas unitarias continuas**, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación. Véase, por ejemplo, las herramientas de prueba *JUnit* orientada a Java, *DUnit* orientada a *Delphi*, *NUnit* para la plataforma .NET o *PHPUnit* para PHP. Estas tres últimas inspiradas en *JUnit*, la cual, a su vez, se inspiró en SUnit, el primer framework orientado a realizar test, realizado para el lenguaje de programación *Smalltalk* (17).
- **Programación en parejas:** se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera (el código es revisado y discutido mientras se escribe) es más importante que la posible pérdida de productividad inmediata (17).
- Frecuente **integración del equipo de programación con el cliente** o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo (17).
- **Corrección de todos los errores** antes de añadir nueva funcionalidad. Hacer entregas frecuentes (17).
- **Refactorización del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo (17).
- **Propiedad del código compartida:** en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados (17).

- **Simplicidad en el código:** es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo (17).

La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más simple es el sistema, menos tendrá que comunicar sobre éste, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

Faces de la metodología XP

La metodología XP consta de cuatro fases y un conjunto de técnicas que se describen a continuación:

Planificación: Las técnicas utilizadas en esta fase inicial son las historias de usuario (HU), planes de entrega e iteración. Inicialmente se escriben las HU y se realizan las estimaciones para cada una de estas. Las HU deben ser materializadas en el proceso de desarrollo de acuerdo a su orden, fecha, valor y riesgo estimados. Las HU son utilizadas para realizar estimaciones en la planificación de las entregas, son usadas en lugar de documentos de requerimientos, son escritas por el cliente en términos del cliente y guían la creación de pruebas de aceptación. Debe tenerse en cuenta la posibilidad de recuperarse ante una sobre carga, el cambio de valor de una HU, introducir una nueva HU, dividir una HU y re-estimar. Los planes de entrega surgen en las reuniones de planificación de liberaciones, se utilizan para crear planes de iteraciones, y están sujetos a decisiones técnicas por parte del personal técnico y decisiones de negocio por parte del personal de negocio (18).

Diseño: La fase de diseño debe caracterizarse por utilizar una técnica simple, sin complejidad innecesaria, se debe cuidar de no añadir funcionalidades sin antes ser analizadas, esto puede provocar atrasos y malgasto de recursos. Las técnicas a utilizar poden ser metáforas o tarjetas CRC (Clases, Responsabilidades y Colaboradores), estas últimas son las más recomendadas pues sirven para diseñar el sistema entre todo el equipo y permiten reducir el modo de pensar procedural. La refactorización desempeña un papel importante en la reestructuración del código. Se utiliza con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo, hacerlo más flexible para facilitar posteriores cambios y ahorrar tiempo e incrementar la calidad.

La selección de un adecuado patrón arquitectónico y de patrones de diseño facilita y guían la siguiente fase de desarrollo (19).

Desarrollo: En esta fase el cliente siempre está disponible y es el que conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio. La comunicación oral es más efectiva que la escrita, ya que ésta última toma mucho tiempo en generarse y puede tener más riesgo de ser mal interpretada. Durante la fase de desarrollo se determina, dada la prioridad, el orden en que las HU, escritas por el cliente con ayuda del desarrollador, serán incluidas en una determinada iteración; esta técnica es conocida como plan de *releases* o plan de proyecto. Los estándares de programación o estándares de codificación como también se les conoce mantienen el código legible y organizado, facilitando su entendimiento y los cambios que pudieran efectuarse sobre él. La integración secuencial en esta fase garantiza que solo una persona pueda integrar, probar y liberar cambios al repositorio de código, esto entre otras ventajas permite que la última versión esté consistentemente identificada. Por otro lado, la integración continua posibilita que cada pieza de código pueda ser integrada en el sistema una vez que esté lista, así el sistema puede llegar a ser integrado y construido varias veces en un mismo día, evita o detecta antes los problemas de compatibilidad que pudieran ocurrir. Durante la fase de codificación se recomienda trabajar solo 40 horas máximas a la semana dado que el trabajo extra desmotiva al equipo (19).

La metodología define que la fase de pruebas esté presente en cada una de las fases anteriormente mencionadas. Las pruebas son vistas y realizadas como un proceso más de cada fase y debe estar presente en todo momento (19).

Roles que contiene la metodología XP y responsabilidades

Según Roger S. Pressman[20] En la 7ma Edición del Libro Ingeniería de *Software*: Un Enfoque Práctico, la metodología XP contiene los siguientes roles y responsabilidades:

El **programador** escribe las pruebas unitarias y produce el código del sistema. Define las tareas que conlleva cada historia de usuario, y estima el tiempo que requerirá cada una.

El **cliente** escribe las historias de usuario y las pruebas funcionales para validar su implementación. Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

[20] Roger S. Pressman: Ingeniero de *software*, profesor, consultor y autor de productos centrados en la Ingeniería del *Software*.

El **encargado de pruebas** ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

El **encargado de seguimiento** verifica las estimaciones realizadas, evalúa el progreso de cada iteración, así como la factibilidad de los objetivos con las restricciones de tiempo y recursos presentes. Mantiene contacto directo con el equipo de desarrollo, realizando cambios para lograr los objetivos de cada iteración.

El **entrenador** es responsable del proceso global. Experto en XP, provee de las guías a los miembros del equipo para que se apliquen las prácticas XP y se siga el proceso correctamente. Determina la tecnología y metodologías a usar por el equipo de desarrollo.

El **gestor** es el dueño del equipo y sus problemas. Experto en tecnología y labores de gestión. Construye el plantel del equipo, obtiene los recursos necesarios y maneja los problemas que se generan. Administra a su vez las reuniones (planes de iteración, agenda de compromisos, etc.). No le dice al grupo lo que tiene que hacer, cuando hacerlo, ni verifica el avance de las tareas.

Es criterio del investigador que la metodología XP se ajusta a las características del sistema a implementar teniendo en cuenta la solicitud del cliente, ya que el equipo de desarrollo está conformado por una única persona y la metodología en sus principios básicos plantea la programación en equipos pequeños con pocos roles. Otro argumento es que existe comunicación directa con el cliente por lo que va a estar en condiciones de contestar rápida y correctamente a cualquier pregunta por parte del resto del equipo de desarrollo, de forma que no se atrase la toma de decisiones. Es abierta a los cambios y genera poca documentación lo que hace la entrega del *software* menos complicada y más satisfactoria tanto para el cliente como para el equipo de entrega.

1.5 Fundamentación de las herramientas y sus lenguajes

1.5.1 Herramienta para el modelado

Visual Paradigm para UML 8.0

Es una herramienta multiplataforma de modelado UML la cual permite dibujar todos los tipos de diagramas de clases, genera código desde diagramas, realiza ingeniería inversa e informes en varios formatos. Esta herramienta presenta licencia gratuita y comercial (14).

Algunas de las ventajas que lo identifican son (14):

- Navegación entre código y el modelo.
- Generador de documentación y reportes *UML/PDF/HTML/MS Word*[21].
- Demanda en tiempo real y sincronización de código fuente.
- Entorno de modelado visual.
- Diagramas de diseño automático.
- Análisis de texto y soporte de tarjeta CRC.
- Proporciona soporte a varios lenguajes en generación de código e ingeniería inversa a través de plataformas java.

Visual Paradigm para *UML* 8.0 es la herramienta de modelado a utilizar, que soporta los diagramas a modelar en el ciclo de vida completo del *software*.

Lenguaje para el modelado

El Lenguaje Unificado de Modelado (UML) define un conjunto de notaciones y diagramas estandarizados para modelar sistemas orientados a objetos, además de describir la semántica de lo que estos significan (20).

UML puede ser empleado para modelar distintos tipos de sistemas y se usa para especificar, visualizar, construir y documentar artefactos de un sistema de *software*. Además, UML tiene varias ventajas, entre las cuales es posible citar (20):

- Se puede usar para diferentes tipos de sistemas.
- Independencia de la arquitectura o el lenguaje seleccionado para la realización.
- Incorporación de mejores prácticas a nivel internacional.
- Cuenta con un amplio apoyo entre organizaciones e instituciones.
- Define un "meta-modelo" en donde un diagrama define la sintaxis de la notación UML.

1.5.2 Herramienta para el desarrollo

IDE *NetBeans* 8.0

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. *Sun MicroSystems*[22] fundó el proyecto de código abierto *NetBeans* en junio de 2000 y

[21] *MS Word:* Es una aplicación informática orientada al procesamiento de textos.
[22] *Sun Microsystems*: Es una empresa informática adquirida por *Oracle Corporation*, anteriormente parte de *Silicon Valley*, fabricante de semiconductores y *software*.

continúa siendo el patrocinador principal de los proyectos (Actualmente *Sun Microsystems* es administrado por *Oracle Corporation*[23]).

La plataforma *NetBeans* permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de *Software* llamados módulos. Un módulo es un archivo Java que contiene clases de *Java* escritas para interactuar con las *APIs* de *NetBeans* y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma *NetBeans* pueden ser extendidas fácilmente por otros desarrolladores de *software* (21).

IDE NetBeans es libre, de código abierto, multiplataforma con soporte integrado para el lenguaje de programación *Java*. Incluye además compatibilidad con diferentes aplicaciones y lenguajes entre ellos *Java, C#, C/C++ Aplications, PHP and HTML5 Aplications, Java GUI Aplications* (22).

Características principales

Seguidamente se fundamentan algunas de las características que hacen del *Netbeans* una potente herramienta para el desarrollo de aplicaciones (21):

- Suele dar soporte a casi todas las novedades en el lenguaje Java. Cualquier *preview*[24] del lenguaje es rápidamente soportada por *Netbeans*.
- Asistentes para la creación y configuración de distintos proyectos, incluida la elección de algunos *frameworks*.
- Buen editor de código, multilenguaje, con el habitual coloreado y sugerencias de código, acceso a clases pinchando en el código, control de versiones, localización de ubicación de la clase actual, comprobaciones sintácticas y semánticas, plantillas de código, *coding tips*, herramientas de refactorización, etc. También hay tecnologías donde se utiliza pulsar y arrastrar para incluir componentes en el código.
- Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, y estructurando la visualización de manera ordenada, lo que ayuda en el trabajo diario. Una vez dentro del cuerpo de una clase Java, por poner un ejemplo, se muestran distintas ventanas con el código, su localización

[23] *Oracle Corporation*: Es una de las mayores compañías de *software* del mundo. Sus productos van desde bases de datos (*Oracle*) hasta sistemas de gestión. Cuenta, además, con herramientas propias de desarrollo para realizar potentes aplicaciones.

[24] *preview:* Se refiere *a* actualización.

en el proyecto, una lista de los métodos y propiedades (ordenadas alfabéticamente), también hay una vista que presenta las jerarquías que tiene la clase y otras muchas opciones. Por supuesto personalizable según el gusto de cada usuario.

- Herramientas para depurado de errores: el *debugger* que incluye el IDE es bastante útil para encontrar dónde fallan las cosas. Permite definir puntos de ruptura en la línea de código que el usuario desee, monitorizar en tiempo real los valores de propiedades y variables, etc. Incluso se puede usar el *debugger* en caliente, permitiendo la interacción con el mismo mientras el proceso esté o no en ejecución.
- Optimización de código: por su parte el *Profiler* ayuda a optimizar las aplicaciones permitiendo que se ejecuten más rápido y con el mínimo uso de memoria. Puede ser configurado a gusto del usuario, aunque por defecto, ofrece opciones bastante útiles. Lo importante es que permite ver el comportamiento de la aplicación y obtener indicadores e información de cómo y cuantos recursos consume, cuantos objetos se crean, también permite obtener capturas del estado del sistema en diferentes momentos (*Snapshots*) y compararlos entre sí.
- Acceso a base de datos: desde el propio *Netbeans* es posible la conexión a distintos sistemas gestores de bases de datos, como pueden ser *Oracle*, *MySql* y demás, y ver las tablas, realizar consultas y modificaciones, y todo ello integrado en el propio IDE.
- Se integra con diversos servidores de aplicaciones, de tal manera que permite la gestión de los mismos desde el propio IDE: inicio, parada, arranque en modo *debug*, despliegues. Entre otros se pueden utilizar: *Apache Tomcat*, *GlassFish*, *JBoss*, *WebLogic*, *Sailfin*, *Sun Java System Application Server*, etc.

Lenguaje para el desarrollo

Java

Java es un lenguaje de programación orientado a objetos desarrollado por *Sun Microsystems* a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de lenguaje de programación C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por *Sun Microsystems* en 1995. Desde entonces, *Sun* ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java *Community*

Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de *Sun*, algunas incluso bajo licencias de *software* libre (23).

El lenguaje es inusual porque los programas Java son tanto compilados como interpretados. La compilación, que ocurre una vez por programa, traduce el código Java a un lenguaje intermedio llamado Java *bytecode*. El *bytecode*, por su turno, es analizado y ejecutado (interpretado) por Java *Virtual Machine* (JVM), un traductor entre el idioma, el SO subyacente y el *hardware*. Todas las implementaciones del lenguaje de programación deben emular JVM, para permitir que los programas Java se ejecuten en cualquier sistema que tenga una versión de JVM (24).

Características del lenguaje

Java es uno de los lenguajes de programación más potentes del mundo, posee muchas características dentro de las que destacan (25):

Simple: Basado en el lenguaje C++, pero donde se eliminan muchas de las características *OOP* que se utilizan esporádicamente y que creaban frecuentes problemas a los programadores.

Orientado al objeto: Java da buen soporte a las técnicas de desarrollo *OOP* y en resumen a la reutilización de componentes de *software*.

Distribuido: Java se ha diseñado para trabajar en ambiente de redes y contienen una gran biblioteca de clases para la utilización del protocolo TCP/IP, incluyendo HTTP y FTP.

Interpretado: El compilador Java traduce cada fichero fuente de clases a código de *bytes* (*Bytecode*), que puede ser interpretado por todas las máquinas que den soporte a un visualizador de que funcione con Java.

Sólido: El código Java no se quiebra fácilmente ante errores de programación.

Seguro: Como Java suele funcionar en ambiente de redes el tema de seguridad debe interesar en sobremanera. Actualmente se está trabajando en cifrar el código.

Arquitectura neutral: El compilador crea códigos de *byte* (*Bytecode*) que se envía al visualizador solicitado y se interpreta en la máquina que posee un intérprete de Java o dispone de un visualizador que funciona con Java.

Portable: Al ser de arquitectura neutral es altamente portable, pero esta característica puede verse de otra manera: Los tipos estándares (*int, float, strig*, etc.) están igualmente implementados en todas las máquinas por lo que las operaciones aritméticas funcionaran igual en todas las máquinas.

Alto desempeño: Al ser código interpretado, la ejecución no es tan rápida como el código compilado para una plataforma particular.

Multihilos: Java puede aplicarse a la realización de aplicaciones en las que ocurra más de una cosa a la vez. Java, apoyándose en un sistema de gestión de eventos basado en el paradigma de condición y monitores C.A.R[25], permite apoyar la conducta en tiempo real y de forma interactiva en programas.

Dinámico: Al contrario que C++ que exige que se compile de nuevo la aplicación al cambiar una clase madre Java utiliza un sistema de interfaces que permite aligerar esta dependencia.

1.6 Conclusiones parciales

De los aspectos tratados en el capítulo se puede concluir que:

- El análisis de los sistemas existentes arrojó que los mismos permiten el control centralizado de los dispositivos de *hardware* y en su mayoría permiten su utilización en aplicaciones *web*.
- El análisis del ambiente de desarrollo permitió establecer la metodología, las herramientas y lenguajes a utilizar en la implementación del sistema de acuerdo a las características solicitadas por el cliente.

Una vez realizado el estudio de los sistemas existente y el análisis del ambiente de desarrollo se determinó la necesidad de crear una solución propia que responda a las características de las entidades cubanas y cumpla con el paradigma de independencia tecnológica por la que aboga el país.

[25] Monitores C.A.R: Monitor creado por los científicos Per Brinch Hansen (informático danés-estadounidense conocido por la teoría de la programación concurrente) y el británico Charles Antony Richard Hoare (conocido sobre todo por la invención, en 1960 de *Quicksort*, que es el algoritmo de ordenamiento más ampliamente utilizado en el mundo).

Capítulo II: Características del Sistema

2.1 Introducción

Luego de haber realizado un estudio del arte e identificado una solución a la necesidad evidenciada en la problemática, se propone el siguiente capítulo en el que se describirán las características principales que posee el sistema propuesto. Se define el modelo conceptual actual, describiendo además cada una de las entidades que intervienen en este modelo en aras de lograr un mayor entendimiento del mismo, se confeccionan las historias de usuario y las tareas de ingeniería asociadas a cada una de ellas. Se explica con detalles la propuesta del sistema, definiendo cada uno de los subsistemas que lo componen, así como un diagrama que provee una vista general de mismo.

2.2 Presentación de la solución

La solución estará conformada por una aplicación que permitirá manejar todos los recursos (estaciones de trabajos y dispositivos) de manera centralizada. Su principal objetivo será proporcionar un mecanismo para poder usar los dispositivos empleados para imprimir documentos y capturar imágenes, además de permitir la interacción con aplicaciones *web*. La ***Ilustración 1*** muestra una abstracción del sistema. Dicha solución implementará servicios de impresión, cámaras, e información de las estaciones de trabajo. En algún momento pueda ser utilizada para implementar llaves o identificadores propios de la estación de trabajo.

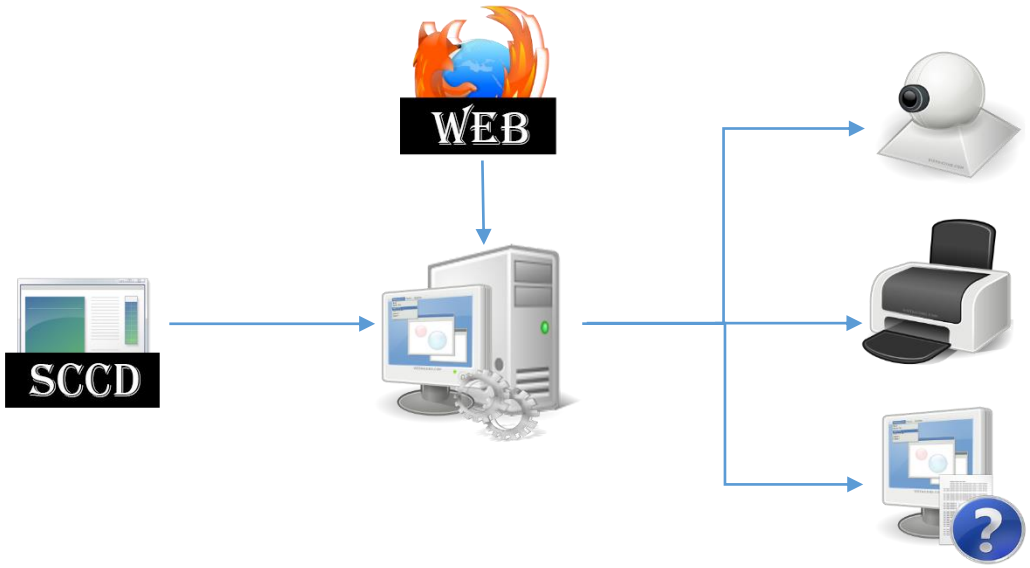

Ilustración 1: Abstracción del sistema. Fuente: Elaboración Propia

Capitulo II

2.3 Modelo Conceptual

No es más que una herramienta de comunicación fundamental que obliga y permite a los desarrolladores pensar formalmente en el problema y a validar la comprensión del mismo. Además, establece un vocabulario propio del problema; y constituye, junto a los requerimientos, la entrada más importante para el diseño (26).

El modelo conceptual, es una representación visual estática que se realiza con el objetivo de lograr una mejor comprensión del contexto para la obtención de requisitos del sistema. Básicamente, de manera visual es un diagrama de clase UML que modela los conceptos básicos asociados al dominio del problema, sus propiedades más importantes y las relaciones que resultan imprescindibles para contextualizar dichos conceptos, a fin de poder brindar elementos que permitan la identificación de los requisitos del sistema. Además, es válido para cualquier metodología de desarrollo de *software* (27).

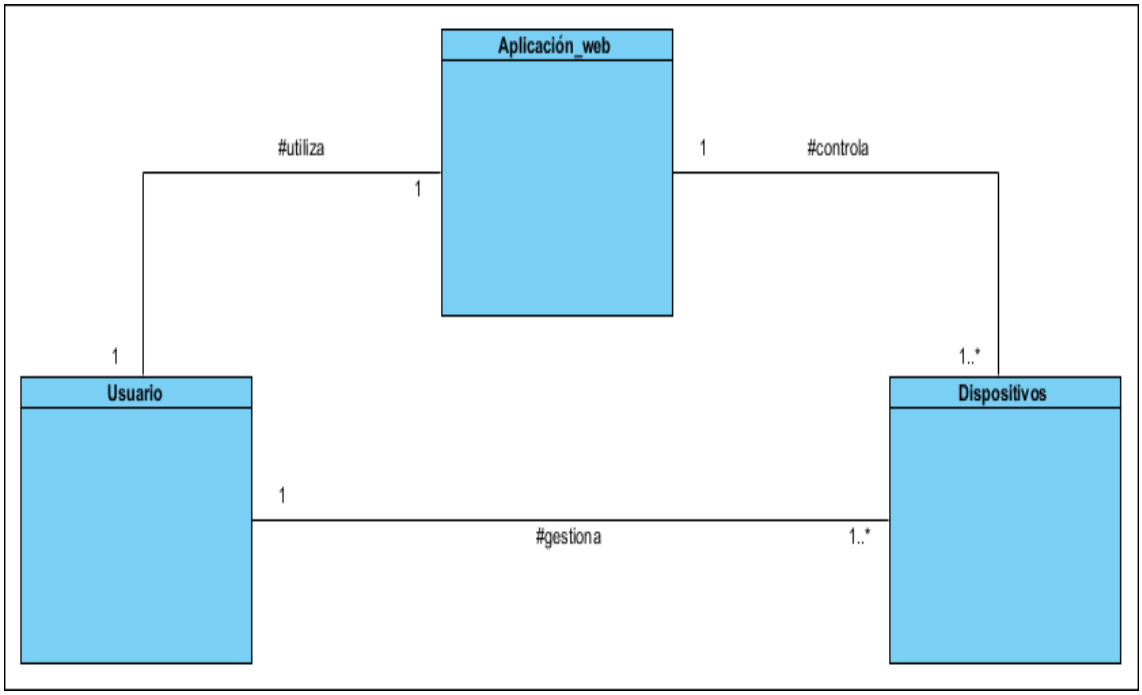

Ilustración 2: Modelo conceptual. Fuente: Elaboración Propia.

2.3.1 Glosario de términos del Modelo Conceptual

Para un mejor entendimiento en este acápite se explicarán cada una de las clases que conforma el Modelo De Dominio. Ver ***Ilustración 2***.

Usuario: Persona encargada de interactuar con el servicio y acceder a sus funcionalidades. Posee un grupo de permisos que lo asocian al uso de un grupo de dispositivos.

Aplicación web: Contenido que se ejecuta dentro del navegador *web*, que está destinado para la interacción con el usuario a través de las funcionalidades que brindan, y que en determinados casos requieren el uso de dispositivos.

Dispositivos: Accesorios de *hardware* que pueden ser accedidos remotamente a través de una aplicación *web* por los usuarios. Estos dispositivos están conformados por impresoras para la impresión de documentos y cámaras *web* para la captura de imágenes.

2.4 Descripción del sistema propuesto

El sistema propuesto es un demonio que estará instalado en las estaciones de trabajo de los clientes sirviendo de intermediario entre las aplicaciones *web* y los dispositivos de *hardware* conectados a las mismas, permitiendo que la aplicación *web* obtenga el control centralizado sobre estos dispositivos. El sistema está compuesto por un *framework JS* el cual facilita esta conexión y tres servicios fundamentales:

- el servicio de impresión, el cual posee una configuración que permite definir qué impresora utilizar, o sea, si utiliza una impresora nombrada o utiliza la impresora por defecto del SO *GNU/Linux*. Este servicio permite la impresión remota de cualquier tipo de documento.
- el servicio de cámaras, el cual posee los sub-servicios, *webcam* (en caso de que la estación de trabajo sea una laptop) y cámara USB para las restantes estaciones de trabajo. Este servicio permite la captura en tiempo real de imágenes.
- el servicio de información de las estaciones de trabajo, este servicio permite obtener la información básica del *hardware* y el *software* de las estaciones de trabajo.

La comunicación entre la aplicación *web* y los dispositivos de *hardware* conectados a las estaciones de trabajo ocurre de la siguiente forma:
La aplicación *web* envía una solicitud a través del protocolo HTTP al sistema del servicio al que desea acceder, ya sea el de la impresora, la cámara *web* o el de información de la estación de trabajo, éste verifica si los servicios están disponibles, en caso de que estén disponibles reenvía dicha petición a la aplicación *web* permitiéndole el acceso y el control centralizado del servicio solicitado, en caso contrario lanza un mensaje de error en el cual describe el motivo por el cual el servicio se encuentra inaccesible en ese momento.

framework JS: básicamente cumple con las mismas funciones y requisitos que el *framework DGMJS* ya que es una adaptación del mismo al nuevo sistema propuesto, o

sea, el Servicio SCCD. El *framework JS* es un componente que cuenta con un grupo de clases que permiten la interacción con los servicios publicados en la estación de trabajo donde se esté ejecutando el SCCD. De esta manera se establece una comunicación entre la aplicación *web* externa y el SCCD lo cual posibilita la interacción con los dispositivos.

Control centralizado: Desde una estación de trabajo el usuario, mediante el uso de aplicaciones *web*, puede tener acceso hacia todas las restantes estaciones de trabajo que conforman la red de la empresa o institución en las cuales estará instalado el SCCD y de esta forma, poder gestionar los servicios que integran dicho sistema, ya sea el de impresión, *webcam* o simplemente obtener la información básica de las estaciones de trabajo sin la necesidad de levantarse de sus puestos de trabajo.
En la siguiente ilustración se muestra la vista general del sistema y la relación que existe entra cada uno de los componentes que integran la solución:

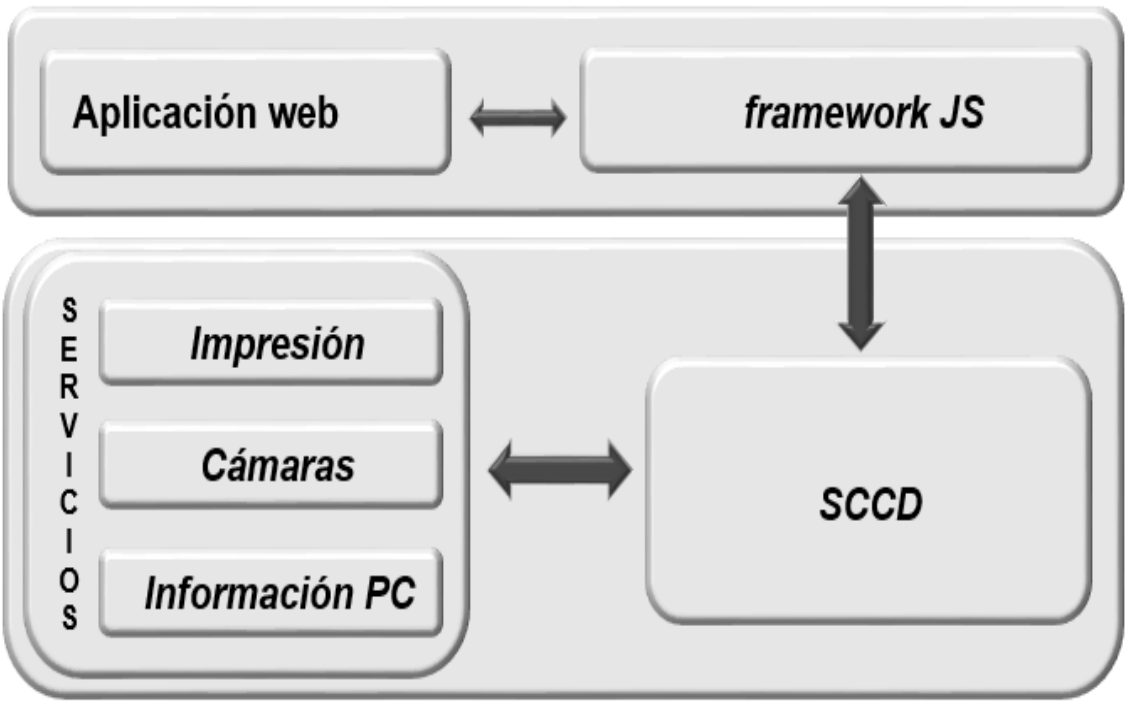

Ilustración 3: Vista general del sistema. Fuente: Elaboración Propia.

2.5 Requisitos del sistema

Un requisito es una circunstancia o condición necesaria para algo (28). En ingeniería de sistemas se emplea el término requisito en un sentido análogo, como una condición necesaria sobre el contenido, forma o funcionalidad de un producto o servicio.

Los requisitos especifican qué es lo que el sistema debe hacer (sus funciones) y sus propiedades esenciales y deseables. La captura de los requisitos tiene como objetivo

principal la comprensión de lo que los clientes y los usuarios esperan que haga el sistema. Un requisito expresa el propósito del sistema sin considerar como se va a implantar. En otras palabras, los requisitos identifican el **qué** del sistema, mientras que el diseño establece el **cómo** del sistema (29).

La captura y el análisis de los requisitos del sistema es una de las fases más importantes para que el proyecto tenga éxito. Como regla de modo empírico, el costo de reparar un error se incrementa en un factor de diez de una fase de desarrollo a la siguiente, por lo tanto, la preparación de una especificación adecuada de requisitos reduce los costos y el riesgo general asociado con el desarrollo (29).

Los requisitos son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos suelen clasificarse en requisitos funcionales o no funcionales (27).

Los funcionales declaran los servicios que debe brindar el sistema, la manera que éste debe reaccionar y funcionar ante una entrada o situación en particular y los no funcionales son las restricciones de los servicios o funciones ofrecidas por el sistema (27).

2.5.1 Definición de los requisitos funcionales del sistema

Los requisitos funcionales declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. Los requisitos funcionales de un sistema describen lo que el sistema debe hacer. Estos requisitos dependen del tipo de *software* que se desarrolle, de los posibles usuarios del *software* y del enfoque general tomado por la organización al redactar requisitos. Cuando se expresan como requisitos del usuario, habitualmente se describen de una forma bastante abstracta. Sin embargo, los requisitos funcionales del sistema describen con detalle la función de éste, sus entradas y salidas, excepciones, etc. (30). A continuación de describen los requisitos funcionales definidos para el SCCD:

Rf1.Obtener la información del *hardware* de las estaciones de trabajo.

Rf2.Gestionar la *webcam* de las estaciones de trabajo.

- Rf2.1. Activar la *webcam*.
- Rf2.2. Capturar la imagen.
- Rf2.3. Guardar la imagen.

Rf2.4. Desactivar la *webcam*.

Rf3.Gestionar las cámaras conectadas (por *USB*) a las estaciones de trabajo.

Rf3.1. Capturar la imagen que reproduce la cámara.

Rf3.2. Guardar la imagen que reproduce la cámara.

Rf4. Gestionar las impresoras conectadas a las estaciones de trabajo.

Rf4.1. Activar la impresora.

Rf4.2. Imprimir el documento.

Rf4.3. Desactivar la impresora.

Rf5.Actualizar el servicio.

2.5.2 Definición de los requisitos no funcionales del sistema

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema (30).

Los requisitos no funcionales, como su nombre indica, son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (30).

Los requisitos no funcionales rara vez se asocian con características particulares del sistema. Más bien, estos requisitos especifican o restringen las propiedades emergentes del sistema. Por lo tanto, pueden especificar el rendimiento del sistema, la protección, la disponibilidad, y otras propiedades emergentes. Esto significa que a menudo son más críticos que los requisitos funcionales particulares. Los usuarios del sistema normalmente pueden encontrar formas de trabajar alrededor de una función del sistema que realmente no cumple sus necesidades. Sin embargo, el incumplimiento de un requisito no funcional puede significar que el sistema entero sea inutilizable (30).

Los requisitos no funcionales se clasifican en distintas categorías; requisitos de *software*, *hardware*, restricciones del diseño, implementación, apariencia, usabilidad,

seguridad y soporte son algunas de ellas (31). A continuación, se presenta el conjunto de requisitos no funcionales definidos para el SCCD:

Requisitos de apariencia

RnF 1. El sistema debe contar con una interfaz accesible e intuitiva para controlar y mostrar el estado de los servicios al usuario, así como para realizar las configuraciones necesarias.

RnF 2. Los elementos del sistema contendrán claro y bien estructurados los datos, y al mismo tiempo permitirán la interpretación correcta e inequívoca de la información.

Requisitos de *software*

RnF 3. Sistema Operativo: *GNU/Linux*.

RnF 4. Servidor *GalssFish 4.0*

Requisitos de seguridad

RnF 5. Garantizar tratamiento de excepciones.

RnF 6. Garantizar el control de acceso a las aplicaciones.

Requisitos de *hardware*

RnF 7. Las estaciones de trabajo cliente deben tener como mínimo 100 Mb de espacio en el Disco Duro, 128 Mb de RAM y procesador *Pentium IV* o superior.

Requisitos de diseño e implementación

RnF 8. El sistema debe implementarse usando el lenguaje *Java*.

RnF 9. El sistema debe desarrollarse usando el *IDE NetBeans 8.0.*

2.6 Historias de usuario

La metodología XP utiliza la técnica de historias de usuario (HU) para especificar los requisitos del *software*. Las mismas describen brevemente las características que el sistema debe poseer. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla (32).

Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario. Su nivel de detalle debe ser el mínimo posible, de manera que permita hacerse una ligera idea de cuánto costará implementar el sistema (32).

Durante el análisis en la fase de exploración fueron identificadas once HU, cada una de ellas corresponden a las diferentes funcionalidades, además proporcionan una idea al desarrollador de cómo debe ser su posterior implementación (32). Seguidamente se

describen tres de las historias de usuario más significativas. Para consulta las restantes HU, ver ***Anexo 1***.

Historia de Usuario	
Numero: HU_1	**Usuario:** Sistema
Nombre de la historia: Gestionar la *Webcam* de las Estaciones de Trabajo.	
Prioridad en negocio: Muy Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 1
Responsable: Joseph Michael Cordero Korolev	
Descripción: La HU se inicia cuando el administrador accede al f*ramework JS* utilizando el navegador *web*. En el mismo puede gestionar la *webcam* de la estación de trabajo utilizando los siguientes botones: 1. Activar: Permite activar la *webcam* de la estación de trabajo. 2. Capturar: Permite capturar en tiempo real la imagen que esté reproduciendo la *webcam*. 3. Guardar: Permite guardar la imagen. 4. Desactivar: Permite desactivar la *webcam*.	
Observaciones: 1. Debe estar disponible el servicio *web*.	

Tabla 2: HU_1. Gestionar la webcam de las estaciones de trabajo. Fuente: Elaboración Propia.

Historia de Usuario	
Numero: HU_3	**Usuario:** Administrador
Nombre de la historia: Gestionar las Impresoras Conectadas a las Estaciones de Trabajo.	
Prioridad en negocio: Muy Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 3	**Iteración asignada:** 1
Responsable: Joseph Michael Cordero Korolev	
Descripción: La HU se inicia cuando el administrador accede al *framework JS* utilizando el navegador *web*. En el mismo puede gestionar la impresora que esté conectada a la estación de trabajo utilizando los siguientes botones: 1. Activar: Permite activar la impresora. 2. Imprimir: Primero verifica si tiene hojas, si la tiene inicia la impresión, si no muestra un mensaje de error diciendo que la impresora no tiene hojas. 3. Desactivar: Permite desactivar la impresora.	
Observaciones: 1. Debe estar disponible el servicio web.	

2. La impresora debe estar conectada a la estación de trabajo.

Tabla 3: HU_2. Gestionar las impresoras conectadas a las estaciones de trabajo. Fuente: Elaboración Propia.

Historia de Usuario	
Numero: HU_4	**Usuario:** Administrador
Nombre de la historia: Obtener la Información del *Hardware* de las estaciones de Trabajo.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada:** 2
Responsable: Joseph Michael Cordero Korolev	
Descripción: La UH inicia cuando el administrador desea obtener la información de la estación de trabajo, para ello accede al *framework* JS a través del navegador *web* y presiona el botón de InfoPC.	
Observaciones: 1. El servicio debe estar iniciado. 2. La PC debe estar conectada a la red.	

Tabla 4:: HU_4. Obtener la información del hardware de las estaciones de trabajo. Fuente: Elaboración Propia

2.6.1 Estimación del esfuerzo por historia de usuario

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las HU generalmente valen de 1 a 3 puntos.

Historias de Usuario	Puntos Estimados
Obtener la información del *hardware* de las estaciones de trabajo.	2
Gestionar la *webcam* de las estaciones de trabajo.	3
Gestionar las cámaras conectadas (por *USB*) a las estaciones de trabajo.	3
Gestionar las impresoras conectadas a las estaciones de trabajo.	3
Actualizar el servicio.	2
Total	13

Tabla 5: Estimación del esfuerzo por historia de usuario. Fuente: Elaboración Propia.

2.7 Planificación

La planeación efectiva de un proyecto de *software* depende de la planeación detallada de su avance, anticipando problemas que puedan surgir y preparando con anticipación

soluciones tentativas a ellos. Se supondrá que el administrador del proyecto es responsable de la planeación desde la definición de requisitos hasta la entrega del sistema terminado (33).

En esta fase el cliente establece la prioridad de cada historia de usuario, y respectivamente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias.

Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

2.7.1 Plan de iteración

Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas específicas de programación. Asimismo, para cada historia de usuario se establecen las pruebas de aceptación. Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir (34).

Iteración	Orden de las HU	Duración
#1	1. Gestionar la *webcam* de las estaciones de trabajo. 2. Gestionar las cámaras conectadas (por USB) a las estaciones de trabajo. 3. Gestionar las impresoras conectadas a las estaciones de trabajo.	3 semanas

#2	1. Obtener la información del *hardware* de las estaciones de trabajo.	5 semanas
#3	1. Actualizar el servicio.	5 semanas

Tabla 6: Plan de iteraciones. Fuente: Elaboración Propia.

Primera Iteración

El objetivo principal de esta iteración es la implementación de las historias de usuarios seleccionadas de mayor prioridad, de esta forma se obtienen las primeras funcionalidades tales como la manejar los servicios *web*, manejar los dispositivos remotos, aceptar el manejo de dispositivos remotamente y gestionar los servicios *web* a usar en el servicio local para el manejo de dispositivos.

Segunda Iteración

En esta iteración se implementan las historias de usuario de prioridad alta correspondientes con aceptar nuevos controladores de dispositivos, gestionar los dispositivos a usar en el servidor *web*, así como validar el acceso a los servicios *web*, Además se corrigen los errores de las HU de la iteración anterior. De esta forma se tiene la segunda versión del producto.

Tercera Iteración

En esta iteración se implementan las HU de prioridad media, de esta forma se obtiene la versión final del producto. A esta versión se le realizarán pruebas para la evaluación de su comportamiento y rendimiento.

2.7.2 Plan de entregas (Releases)

Un plan *releases* o plan de proyecto es un conjunto de historias de usuario agrupadas por versiones del producto, también conocido como plan de entregas. Le posibilita al dueño de producto y al equipo decidir cuánto se debe desarrollar y cuánto tiempo se tardará antes de tener un producto entregable. Sirve para realizar un análisis del progreso.

Release	Descripción de la Iteración	Orden de la HU a implementar	Duración total
1ra Iteración	Historias de usuario de prioridad Alta y Muy Alta.	Gestionar las cámaras conectadas (por USB) a las estaciones de trabajo. Gestionar la *webcam* de las estaciones de trabajo. Gestionar las impresoras conectadas a las estaciones de trabajo. Obtener la información del *hardware* de las estaciones de trabajo.	8 Semanas
2da Iteración	Historias de usuario de prioridad Media y Baja.	Actualizar el servicio.	5 semanas

Tabla 7: Plan de Release. Fuente: Elaboración Propia.

2.8 Diseño

El papel del diseño en el ciclo de vida del *software* es adquirir conocimiento de su funcionamiento. Este constituye el punto de partida para las actividades de implementación, dando soporte a los requisitos funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables y sistemas operativos, que debe poseer la aplicación (35).

2.8.1 Tarjetas de Clase, Responsabilidad y Colaboración (CRC)

Las Tarjetas CRC (Clase, Responsabilidad y Colaboración) constituyen uno de los artefactos de la metodología XP que guía el proceso de desarrollo de la solución propuesta. Sirven para diseñar el sistema entre todo el equipo. Permiten reducir el modo de pensar procedural y apreciar la tecnología de objetos (27).

A continuación, se muestran las tarjetas CRC pertenecientes a las clases del SCCD:

InfoPC	
Responsabilidades	**Colaboradores**
Se encarga de mostrar la información básica de las estaciones de trabajo donde se encuentre instalado el servicio.	----------

Tabla 8: Tarjeta CRC_1. Información de las estaciones de trabajo. Fuente: Elaboración Propia.

Webcam	
Responsabilidades	**Colaboradores**
Es la que permite el control sobre la *webcam* de las estaciones de trabajo donde se encuentre instalado el servicio.	----------

Tabla 9: Tarjeta CRC_2. Webcam. Fuente: Elaboración Propia.

Impresora	
Responsabilidades	**Colaboradores**
Es la que permite gestionar las impresoras conectadas a las estaciones de trabajo donde se encuentre instalado el servicio.	----------

Tabla 10: Tarjeta CR_4: Impresora. Fuente: Elaboración Propia.

2.8.2 Descripción de la arquitectura

Los patrones arquitectónicos se utilizan para expresar una estructura de organización base o esquema para un *software*. Lo cual proporciona un conjunto de sub-sistemas predefinidos, puntualizando sus responsabilidades, reglas y directrices que determinan la organización, comunicación, interacción y relaciones entre ellos.

Los patrones arquitectónicos heredan mucha de la terminología y conceptos de patrones de diseño, pero se centran en proporcionar modelos y métodos re-utilizables específicamente para la arquitectura general de los sistemas de información. Quiere decir que a diferencia de los patrones de diseño estas son plantillas incompletas y no se pueden aplicar directamente al código con modificaciones meramente contextuales (27).

Para el desarrollo del sistema para la comunicación con dispositivos desde aplicaciones *web* en *GNU/Linux* se utilizó la arquitectura n-capas, la cual está basada en una distribución jerárquica de roles y responsabilidades, propiciando una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de interacción con otras capas, y las responsabilidades la funcionalidad que implementan. Ver ***Ilustración 4.***

Capa de Presentación

Es la que se encarga de que el sistema interactúe con el usuario y viceversa, muestra el sistema al usuario, le presenta la información y obtiene la información del usuario en un mínimo de proceso. Es conocida también como interfaz gráfica y debe tener la

característica de ser amigable, o sea, entendible y fácil de usar para el usuario. Esta capa se comunica únicamente con la capa intermedia o de negocio.

Capa de Negocio

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

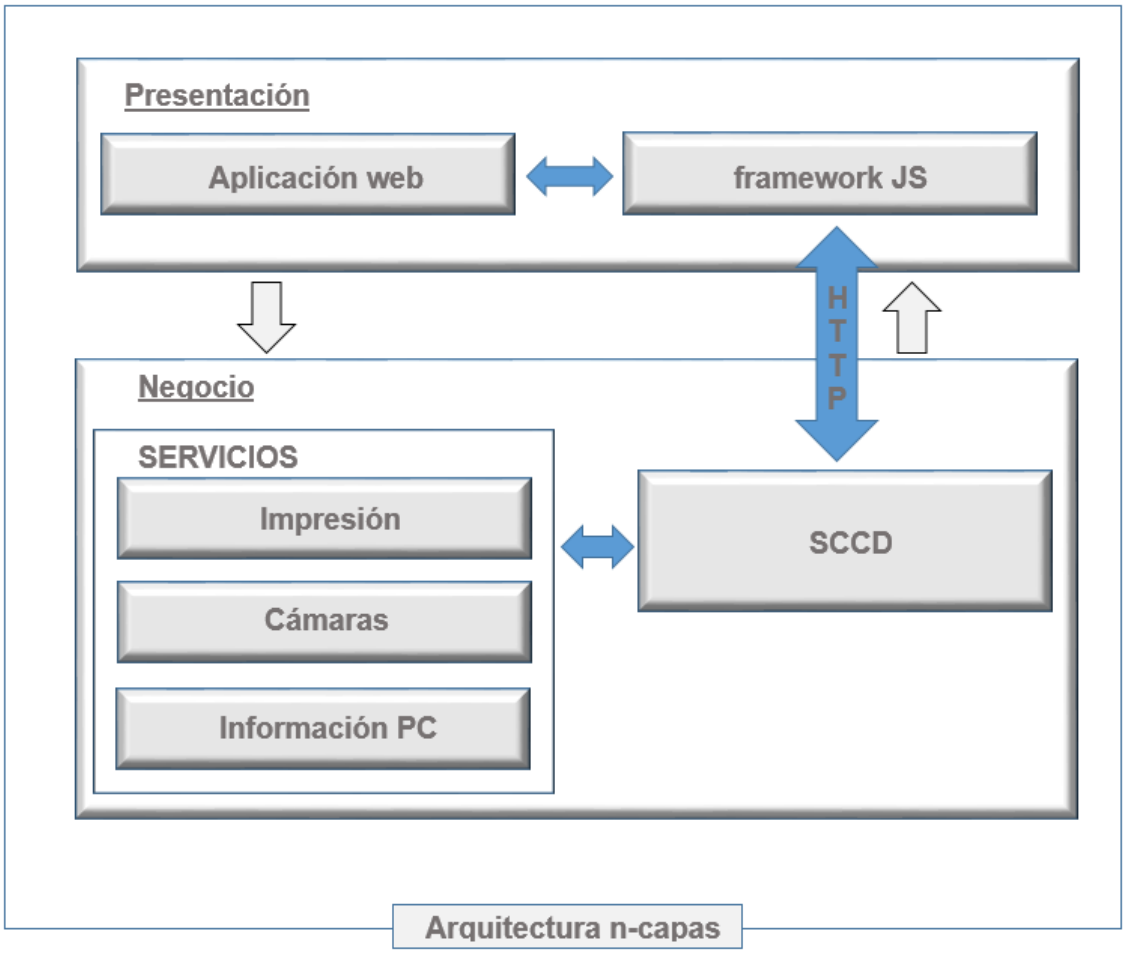

Ilustración 4: Representación de Arquitectura n-capas. Fuente: Elaboración Propia.

2.8.3 Patrones de diseño

Al iniciar el desarrollo de un sistema de *software* es de vital importancia identificar problemas comunes en el diseño del producto para aplicar soluciones existentes que se apliquen a estos. Por tal motivo, uno de los aspectos a considerar en desarrollo de una aplicación son los patrones de diseño de *software* que puedan utilizarse.

Capitulo II

Los patrones de diseño son **soluciones para problemas típicos y recurrentes** que se pueden encontrar a la hora de desarrollar una aplicación. Aunque la aplicación sea única, tendrá partes comunes con otras aplicaciones: acceso a datos, creación de objetos, operaciones entre sistemas etc. En lugar de reinventar, se puede solucionar problemas utilizando algún patrón, ya que son soluciones probadas y documentadas (36).

Patrones generales de *software* para asignar responsabilidades

En el diseño orientado a objetos, es muy importante asignar las responsabilidades de forma correcta. Para ello se utilizan los patrones GRASP, los cuales promueven buenos principios para la realización de dicha tarea. Para el diseño de la propuesta que se desea implementar fueron empleados los siguientes patrones:

Bajo acoplamiento: el diseño del sistema consta de este patrón pues cada clase depende lo menos posible de otra, de manera tal que una de ellas solo recurre a otra en caso de que exista referencia dentro de sus atributos, lo cual permite que el sistema sea mucho más robusto y de fácil mantenimiento (19). Se evidencia el uso de este patrón en las clases *InfoPC_Client*, *Printer_Client* y *WEB_CAM_Client* las cuales no dependen una de la otra ya que son independientes.

Creador: este patrón es el responsable de asignarle a la clase B la responsabilidad de crear una instancia de clase A. B es un creador de los objetos A. Un ejemplo de lo antes mencionado es la clase *DGM_Configuration* la cual crea instancias de las clases *InfoPCSettings*, *PrintSettings*, *WEBCAMSettings*, para utilizar los métodos que éstas poseen (17).

```
import dgm_free.Info_PC.InfoPCSettings;
import dgm_free.Printer.PrintSettings;
import dgm_free.Web Cam.WEBCAMSettings;

public final class DGM_Configurations extends javax.swing.JFrame
{

   public DGM_Configurations()
   {
      initComponents();
      this.setLocationRelativeTo(this);
      jComboBox2.setModel(new DefaultComboBoxModel(new String[] {"No configurable"}));
      jComboBox1.setModel(new DefaultComboBoxModel(new String[] {"Genérico"}));
      jComboBox1.setEnabled(false);
      jComboBox2.setEnabled(false);
      jComboBox5.setModel(new DefaultComboBoxModel(new String[] {"WebCam", "Cannon NX110"}));
      jComboBox4.setModel(new DefaultComboBoxModel(new String[] {"No configurable"}));
      jComboBox4.setEnabled(false);
      jComboBox6.setModel(new DefaultComboBoxModel(new String[] {"PDF", "Impresora nombrada"}));
      jComboBox7.setModel(new DefaultComboBoxModel(new String[] {"Genérico"}));
      jComboBox7.setEnabled(false);

      if(InfoPCSettings.Estatus())
      {
         jComboBox3.setSelectedIndex(0);
         jTabbedPane1.setIconAt(0, new javax.swing.ImageIcon(getClass().getResource("/Resourses/punto_verde.png"))
      }
      else
      {
         jComboBox3.setSelectedIndex(1);
         jTabbedPane1.setIconAt(0, new javax.swing.ImageIcon(getClass().getResource("/Resourses/bullet_red.png")))
      }
```

Ilustración 5: Ejemplo de utilización del patrón Creador. Fuente: Elaboración Propia

Subclase: Propone heredar miembros por defecto de una superclase, las clases hijas tienen los atributos y métodos del padre, pero tienen sus propios atributos y funcionalidades (37).

```
46    public class PopupMenu extends Menu {
47
48        private static final String base = "popup";
49        static int nameCounter = 0;
50
51        transient boolean isTrayIconPopup = false;
52
53        static {
54            AWTAccessor.setPopupMenuAccessor(
55                new AWTAccessor.PopupMenuAccessor() {
                    public boolean isTrayIconPopup(PopupMenu popupMenu) {
57                      return popupMenu.isTrayIconPopup;
58                  }
59              });
60      }
61
```

Ilustración 6: Ejemplo de utilización del patrón Subclase. Fuente: Elaboración Propia.

Prototype: Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo. (27).

```
public final class DGM_Configurations extends javax.swing.JFrame
{
    private static DGM_Configurations instance = null;

    .
    .
  public static void construir()
  {
      instance = new DGM_Configurations();
  }

  public static void Show()
  {
      instance.setVisible(true);
  }

  public static void Hide()
  {
      instance.setVisible(false);
  }
```

Ilustración 7: Ejemplo de utilización del patrón Prototype. Fuente: Elaboración Propia.

Singleton: Asegura que solo se cree una instancia de la clase y ofrece un punto global de acceso a esta instancia. El uso de este patrón permite que una clase pueda ser instanciada solo una vez y acceder luego a la instancia (27).

Ejemplo de este patrón presenta la clase **Configuracion_DGM.** Ver ***Ilustración 8***.

```
14  public class ConfiguracionDGM extends javax.swing.JFrame {
15
16      /**
17       * Creates new form ConfiguracionDGM
18       */
19        //singleton
20      private static ConfiguracionDGM instance = null;
21      private DisplayTryIcon DTI = new DisplayTryIcon();
22
23      public ConfiguracionDGM() {
24          initComponents();
25          this.setLocationRelativeTo(this);
26          jList1.setSelectedIndex(0);
27          mostrarInfoPC();
28      }
29  }
30
```

Ilustración 8: Ejemplo de utilización del patrón Singleton. Fuente: Elaboración Propia.

2.9 Conclusiones parciales

De los aspectos tratados en este capítulo se puede concluir que:

- Las necesidades y deficiencias que fueron identificadas para la realización del proceso en cuestión sirvieron de base para realizar la descripción de la propuesta de solución.
- Con la definición de los servicios de la aplicación se garantiza un mecanismo de interacción entre los componentes de la arquitectura del SCCD.
- La obtención de cinco requisitos funcionales a partir de la definición del sistema que se propone, permitió identificar las operaciones que podrán ser realizadas y la determinación de nueve requisitos no funcionales del sistema resultan fundamentales en el logro de una solución acorde a las necesidades del cliente.
- La definición de la arquitectura y los patrones de diseño a utilizar en la realización de la propuesta de solución, propician la uniformidad entre los componentes de *software* que se obtendrán además de una guía en el desarrollo del *software*.

Capítulo III: Implementación y Validación de la Solución Propuesta

3.1 Introducción

En el presente capítulo se describe la implementación del *software*, fase donde se materializa el producto final y se cumple con los requisitos obtenidos al inicio de la investigación. Para ello se definen los estándares de codificación que debe cumplir el desarrollador, se crea el diagrama de componentes, además de presentarse las principales interfaces de usuario de la solución. A partir del código resultante y su funcionamiento se ejecutan las pruebas de aceptación y de camino básico.

3.2 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de *software*, se establece un estándar de codificación para asegurar que los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de *software* incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de *software* creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente (38).

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de *software*. La mantenibilidad del código es la facilidad con que el sistema de *software* puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de *software* en la que todos los programadores influyen especialmente es en la técnica de codificación (38). Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de *software* se convierta en un sistema de *software* fácil de comprender y de mantener (38).

Capítulo III

Las pautas fundamentales definidas para la implementación son las siguientes:

Estructura

- Agrega un único espacio después de cada delimitador coma.

```
String resultado = client.getInfoPC(String.class,"comand_off_service");
```

- No utilizar espacios después de la apertura de un paréntesis y antes del cierre del mismo.

```
String resultado = client.getInfoPC(String.class,"comand_off_service");
```

- Agregar un único espacio alrededor de operadores (==, &&, ...).

```
int valueWC = jComboBox8.getSelectedIndex();
```

- No agregar espacios antes de los paréntesis de apertura de una palabra clave de control (*if, else, for, while*, ...).

```
if("Servicio de WebCam Encendido".equals(resultado))
```

- No agregar espacios al final de las líneas.

```
javax.ws.rs.core.Form form = getQueryOrFormParams(queryParamNames, queryParamValues);
```

- Utiliza llaves para indicar el cuerpo de las estructuras de control sin importar el número de sentencias que éstas contengan.

```
if (paramValues[i] != null){
    form = form.param(paramNames[i], paramValues[i]);
}
```

- Utiliza letras mayúsculas y minúsculas para constantes, con palabras separadas por guiones bajos.

```
webTarget = client.target(BASE_URI).path("infopc");
```

- Declara las propiedades de las clases antes de los métodos.

```
public class InfoPC_Client {
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/InfoPCService/webresources";
```

- Declara los métodos públicos primero, luego los protegidos y finalmente los privados.

```
public class WEBCAMSettings{
    private static boolean encendido=false;
```

Capítulo III

Convención de Nombres

- Utiliza *camelCase* y no guiones bajos, para variables, funciones y nombres de métodos.

```
SubstanceLookAndFeel.setCurrentTheme("org.jvnet.substance.theme.SubstanceEbonyTheme");
```

- Utiliza guiones bajos para definir opciones, argumentos y nombres de parámetros.

```
public class Printer_Client {
```

- Utiliza los *namespace* para todas las clases.

```
dgm_free.Info_PC.InfoPCSettings;
dgm_free.Printer.PrintSettings;
dgm_free.Web_Cam.WEBCAMSettings;
```

- Utiliza *package* como el *namespace* de primer nivel.

```
package dgm_free;
```

- Utiliza caracteres alfanuméricos y guiones bajos para nombres de archivos.

```
tray_icon = new TrayIcon(img);
tray_icon.setImageAutoSize(true);
```

3.3 Interfaces de usuario

En informática, la interfaz de usuario es el espacio por medio del cual se pueden comunicar las personas con las máquinas para que así los usuarios puedan operar y controlar a la máquina, y que ésta a su vez envíe retroalimentación para ayudar al operador a tomar decisiones y realizar tareas (39). A continuación, se muestra la interfaz principal del sistema:

Capítulo III

Ilustración 9: Interfaz de usuario del SCCD. Fuente: Elaboración Propia.

En este caso los servicios de Información PC, *Webcam* e Impresora que brinda el sistema se encuentran detenidos por lo que si un usuario a través de una interfaz *web* solicita cualquiera de estos servicios el sistema devuelve una notificación a través del navegador mediante el cual se esté intentado acceder a los mismos diciendo el servicio está detenido en este momento.

Esta interfaz está compuesta por cinco campos fundamentales:

1. Muestra el controlador que se encuentra activado en ese momento. En este caso el activo es Genérico y en los casos de Web Cam e Impresora:
   - Web Cam: No configurable.
   - Impresora: PDF o impresora nombrada.
2. Muestra la configuración del servicio que esté activo en ese momento. En los casos de Web Cam e Impresora se pueden elegir:
   - Web Cam: Cannon o Webcam.
   - Impresora: No configurable.
3. Muestra el estado de actividad del servicio Si" o "No.

4. Muestra una breve descripción del servicio que se esté configurando en ese momento.
5. Contiene la dirección del Servidor del SCCD.

Estos cambios ocurren al accionar el botón “Aplicar” que se encuentra en la parte inferior derecha de la interfaz.

Para iniciar o detener un servicio se debe seleccionar “Si” o “No” en el campo de Información Básica. A continuación, se mostrarán imágenes del estado de actividad del servicio Información PC:

Ilustración 10: Interfaz de usuario del SCCD. Servicio Información PC iniciado. Fuente: Elaboración Propia.

En esta imagen se puede apreciar que el servicio Información PC está activado ya que el ícono que se encuentra en la parte izquierda de la paleta de la Información de la PC está de color verde si estuviese en rojo significa que el servicio se encuentra detenido.

Al seleccionar “Si” en el campo de Información Básica se muestra un mensaje el cual confirma la activación del servicio al accionar el botón “Aceptar” del mismo. La siguiente imagen corresponde al mensaje informativo de activación del servicio Información PC:

Capítulo III

Ilustración 11: Mensaje informativo de iniciación del servicio InfoPC. Fuente: Elaboración Propia.

Si el usuario desea deshabilitar el servicio solamente debe dirigirse al campo de información Básica y seleccionar la opción “No”. Una vez realizada la elección el sistema mostrará un mensaje informativo el cual confirma la desactivación del servicio al accionar el botón “Aceptar”. La siguiente imagen corresponde al mensaje informativo para detener el servicio Información PC:

Ilustración 12: Mensaje informativo de detención del servicio InfoPC. Fuente: Elaboración Propia.

Luego de realizada la operación el ícono que se encuentra en la parte izquierda de la paleta de la Información de la PC cambia al color rojo lo cual indica que el servicio está detenido.

En la siguiente imagen se pueden observar todos los servicios iniciados:

Capítulo III

Ilustración 13: Interfaz de usuario del SCCD. Todos los servicios iniciados. Fuente: Elaboración Propia.

Para consultar las imágenes de los restantes servicios ver el ***Anexo 4***.

3.4 Pruebas

Las pruebas son un proceso de ejecución de un programa con la intención de descubrir errores. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error aún no detectado (27).

Las pruebas de *software* son aquellos procedimientos que se realizan para verificar la calidad de un producto de *software* y pueden ser aplicadas periódicamente. Estas tienen como objetivo fundamental la identificación de posibles errores, además representan una revisión final de las especificaciones del diseño y de la codificación. La metodología XP establece dos tipos de pruebas: pruebas unitarias y pruebas de aceptación o funcionales (27).

3.4.1 Pruebas de funcionalidad

La prueba de funcionalidad o aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el *software* está listo y que puede

ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el *software* fue construido. Un comentario sobre la secuencia de los niveles de prueba (27).

Estrategia de pruebas funcionales

Roger S. Pressman plantea que una estrategia de prueba del *software* integra los métodos de diseño de caso de pruebas del sistema en una serie bien planeada de pasos que desembocará en la eficaz construcción de *software*. Ésta proporciona un mapa que describe los pasos que se darán como parte de la prueba, indica cuándo se planean y cuándo se dan estos pasos, además de cuánto esfuerzo, tiempo y recursos consumirán. Una estrategia de prueba debe ser lo suficientemente flexible como para promover un enfoque personalizado. Al mismo tiempo, debe ser lo adecuadamente rígida como para promover una planeación razonable y un seguimiento administrativo del avance del producto (27).

La estrategia de prueba para este sistema se enfoca en demostrar que el *software* es funcional y seguro, tiene como objetivo detectar errores que interfieran en el exitoso funcionamiento del mismo y corregirlos. Para realizar las pruebas se utiliza el nivel de prueba de aceptación empleando el tipo de prueba de funcionalidad. El método de pruebas que se empleará es el método basado en caja negra: se refiere a las pruebas que se llevan a cabo sobre la interfaz del *software*. O sea, los casos de prueba pretenden demostrar que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información se mantiene (27). Los casos de prueba de caja negra soportado por la técnica de partición equivalente que es la más utilizada para este método pretenden:

- Demostrar que las funciones del *software* son operativas.
- Que las entradas se aceptan de la forma adecuada y que se produce el resultado correcto.
- Así como que la integración de la información externa (por ejemplo, archivos de datos) se mantiene.

A continuación, se describe el caso de prueba para la HU_1: Gestionar la *Webcam* de las Estaciones de Trabajo, compuesto por un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados para cumplir un objetivo en particular o una función esperada. Siempre es ejecutado como una unidad, desde el comienzo hasta el final.

Capítulo III

Caso de Prueba de Aceptación	
Código: CP1_HU1	**HU_ 1:** Gestionar la *Webcam* de las Estaciones de Trabajo
Responsable de la prueba: Joseph Michael Cordero Korolev	
Descripción: Manejar la *webcam* de la estación de trabajo.	
Condiciones de ejecución: 1. La Estación de trabajo debe estar conectada a la red. 2. El servicio debe estar iniciado.	
Entrada / Pasos de ejecución: 1. Ejecutar el servicio. 2. Capturar la imagen.	
Resultado esperado: Se obtuvo la imagen.	
Evaluación de la prueba: Satisfactoria	

Tabla 11: Caso de prueba de aceptación correspondiente a la HU_1. Fuente: Elaboración Propia.

Caso de Prueba de Aceptación	
Código: CP3_HU3	**HU_ 3:** Gestionar las Impresoras Conectadas a las Estaciones de Trabajo.
Responsable de la prueba: Joseph Michael Cordero Korolev.	
Descripción: Interactuar con un dispositivo remotamente.	
Condiciones de ejecución: 1. La estación de trabajo debe estar conectada a la red. 2. La impresora debe estar conectada a la estación de trabajo. 3. El servicio debe estar iniciado.	
Entrada / Pasos de ejecución: 1. Acceder el servicio. 2. Comprobar si la impresora está conectada a la estación de trabajo. 3. Si está conectada. 3.1. Mandar a imprimir el documento. 4. Si no está conectada. 4.1 Mostrar un mensaje de error.	
Resultado esperado: Se imprimió el documento.	
Evaluación de la prueba: Satisfactoria	

Tabla 12: Caso de prueba de aceptación correspondiente a la HU_3. Fuente: Elaboración Propia

Caso de Prueba de Aceptación	
Código: CP4_HU4	**HU_ 4:** Obtener Información del *Hardware* de las Estaciones de Trabajo.
Responsable de la prueba: Joseph Michael Cordero Korolev.	
Descripción: Se muestra la información del *hardware* las estaciones de trabajo.	
Condiciones de ejecución: 1. La estación de trabajo debe estar conectada a la red. 2. El servicio debe estar iniciado.	
Entrada / Pasos de ejecución: 1. Acceder el servicio. **2.** Obtener la información deseada.	
Resultado esperado: Se obtuvo la información de la estación de trabajo.	
Evaluación de la prueba: Satisfactoria.	

Tabla 13: Caso de prueba de aceptación correspondiente a la HU_4. Fuente: Elaboración Propia.

3.4.2 Resultado de las pruebas de aceptación

A continuación, se identifican los resultados arrojados según las pruebas de aceptación realizadas al sistema, para las cuales se realizaron cuatro iteraciones de pruebas:

- En la primera iteración se obtuvieron como resultado un total de 12 No Conformidades (NC), de ellas 8 Significativas (S) y 4 No Significativas (NS) además se propusieron 4 Recomendaciones (R).
- La segunda iteración de pruebas arrojó como resultado 9 NC, de ellas 6 S y 3 NS y se propusieron 5 R.
- En la tercera iteración de pruebas se obtuvieron como resultado 5 NC, de ellas 3 S y 2 NS y se propusieron 3 R.
- En la cuarta y última iteración no se obtuvieron NC y se propusieron 4 R solamente.

La siguiente gráfica muestra los resultados obtenidos para cada una de las iteraciones antes mencionadas:

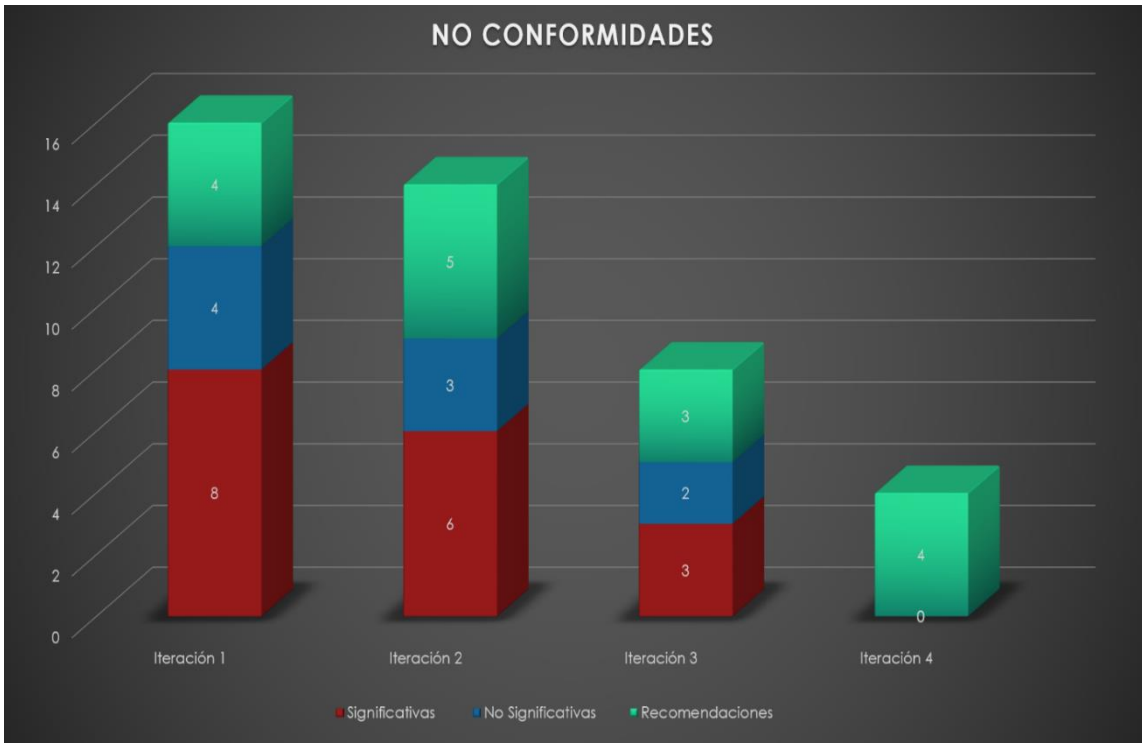

Ilustración 14: Gráfica correspondiente a las no conformidades detectadas. Fuente: Elaboración Propia.

En cada una de las iteraciones fueron resueltas las no conformidades, tanto significativas como no significativas y se valoraron las recomendaciones. En la última iteración no se detectaron no conformidades lo cual demuestra la efectividad con la que fueron realizadas estas pruebas y de esta forma se obtuvo un resultado final satisfactorio del sistema.

3.4.3 Pruebas de caja blanca

Las pruebas de caja blanca, también conocidas como pruebas de caja transparente o pruebas estructurales, se centran en los detalles procedimentales del *software*, por lo que su diseño está ligado al código fuente. Este método intenta garantizar que se ejecutan al menos una vez todos los caminos independientes que presente el módulo y que todas las estructuras de datos internas serán usadas (27).

Para el sistema desarrollado es necesario evidenciar si es un producto de alta calidad, por tanto, es preciso valorar que tan certera ha sido la implementación. Por tanto, se debe aplicar una de las técnicas que comprende las pruebas de caja blanca, en este caso la del camino básico o ruta básica. Para ello es necesario conocer el número de caminos independientes del código al cual se le realizará la prueba, por lo que se construirá un grafo asociado al mismo y se calculará la complejidad ciclomática.

Capítulo III

Estrategia de pruebas de camino básico

Una de las técnicas empleadas para aplicar este criterio de cobertura es la Prueba del Camino Básico. Esta técnica se basa en obtener una medida de la complejidad del diseño procedimental de un programa (o de la lógica del programa). Esta medida es la complejidad ciclomática de McCabe[26], y representa un límite superior para el número de casos de prueba que se deben realizar para asegurar que se ejecuta cada camino del programa. Los pasos a realizar para aplicar esta técnica son:

- Representar el programa en un grafo de flujo.
- Calcular la complejidad ciclomática.
- Determinar el conjunto básico de caminos independientes.
- Derivar los casos de prueba que fuerzan la ejecución de cada camino.

Un Grafo de Flujo está formado por 3 componentes fundamentales que ayudan a su elaboración, comprensión y brinda información para corroborar que el trabajo se está haciendo adecuadamente. Los componentes son:

- **Nodo:** Cada círculo representado se denomina nodo del Grafo de Flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser también que hallan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.
- **Aristas:** Las aristas representan el flujo de control. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.
- **Regiones:** Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran y la cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

[26] Thomas McCabe: En 1976 propuso la métrica nombrada complejidad ciclomática.

Capítulo III

A continuación, se analiza el código y se enumeran las sentencias de código del procedimiento realizado sobre el método **ShowTryIcon ()** el cual se encarga de mostrar el ícono del SCCD en el área de notificación y seguidamente el grafo asociado al mismo:

```
public static void ShowTryIcon()
{
    if(!SystemTray.isSupported())                                    1
    {
        JOptionPane.showMessageDialog(null,"Tu Sistema no
        Soporta Aplicaciones TryIcon", "Reporte de
        Error", JOptionPane.ERROR_MESSAGE);                          2
        return;
    }
        Image img =
        GetImage("Resourses/servicio_iniciado_on.png","Ac
        cess Icon");                                                 3
    if(img == null)
    {
        JOptionPane.showMessageDialog(null,"No se puede
        Leer La Imágen del Icono de la Aplicación",
        "Error de Lectura", JOptionPane.ERROR_MESSAGE);              4
        return;
    }

    tray_icon = new tray_icon();
    tray_icon.setImageAutoSize(true);
                                                                     5
    final SystemTray tray = SystemTray.getSystemTray();

    tray_icon.setPopupMenu(getMenu(tray));
    tray_icon.setToolTip("Sistema para el control
    centralizado de dispositivos v.1.0");

    try
    {
        tray.add(tray_icon);                                         6
    }
    catch(AWTException e)
    {
        JOptionPane.showMessageDialog(null,e.getMessage()
        ,"Excepción",JOptionPane.ERROR_MESSAGE);                     7
        return;
    }
}                                                                    8
```

Ilustración 15: Captura del código del método ShowTryIcon (). Fuente: Elaboración Propia.

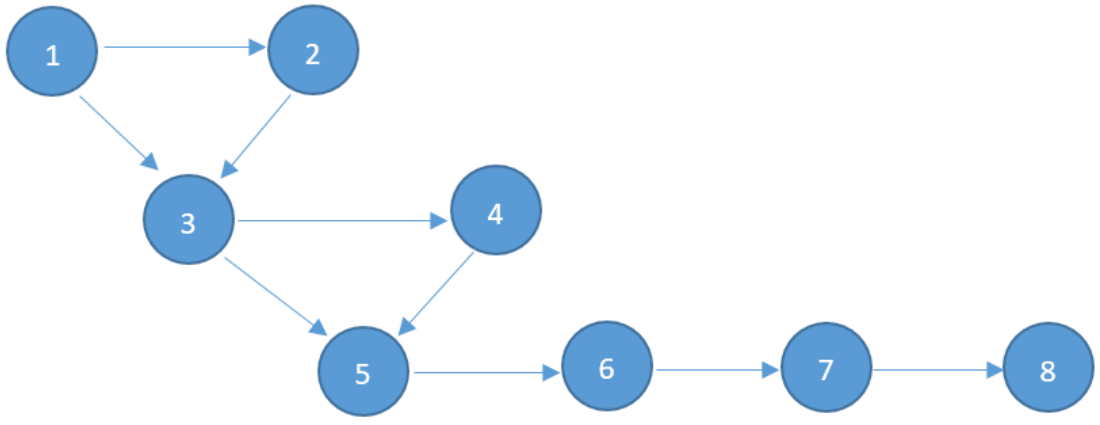

Ilustración 16: Grafo de flujo asociado al método ShowTryIcon (). Fuente: Elaboración Propia.

Capítulo III

Complejidad ciclomática

La complejidad ciclomática es una métrica que proporciona una medición de la complejidad lógica del código, está basado en la teoría de grafos. Es una de las métricas de *software* de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje. El objetivo principal de esta métrica no es contar la cantidad de bucles (*for, while, do*, etc.) sino contar el número de caminos diferentes que existen dentro de un fragmento de código. Por ese motivo algunos mencionan que el nombre más adecuado debería ser el de "Complejidad Condicional". Además, el resultado de este cálculo permite conocer el número de pruebas que se deben hacer para abarcar el 100% del código (40).

Fórmulas para realizar el cálculo de la complejidad ciclomática

1. **V(G) = (A - N) + 2**

Donde "**A**" es la cantidad de aristas y "**N**" es la cantidad de nodos.

V(G) = (9 - 8) + 2

V(G) = 3

2. **V(G) = P + 1**

Siendo "**P**" la cantidad de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V(G) = 2 + 1

V(G) = 3

3. **V(G) = R**

Donde "**R**" representa la cantidad de regiones en el grafo.

V(G) = 3

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es 3, lo cual infiere que existen tres posibles caminos por los cuales puede circular el flujo, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Número	Caminos básicos
1	1-2-3-4-5-6-7-8
2	1-2-3-5-6-7-8
3	1-3-5-6-7-8

Tabla 14: Caminos básicos del flujo asociado al método ShowTryIcon (). Fuente: Elaboración Propia.

Capítulo III

3.5 Conclusiones parciales

Los elementos tratados en este capítulo permitieron concluir que:

- La selección del estándar de codificación permitió una correcta implementación haciendo más vistoso y entendible el código utilizado para el desarrollo del sistema lo cual garantiza el cumplimiento de las reglas y las normas de implementación.
- La etapa de prueba refleja la calidad con que ha sido llevada a cabo la proyección del sistema, permitiendo encontrar y documentar los defectos que pudieron afectar la calidad del *software*.

Se realizaron las pruebas al sistema con resultados satisfactorios que demuestran que la aplicación cuenta con las características y funcionalidades para las que fue concebido.

Conclusiones Generales

La investigación desarrollada y los resultados obtenidos permiten al autor plantear las siguientes conclusiones:

- El estudio de los principales conceptos y sistemas existentes a nivel mundial demostró que estas herramientas, aunque poseen características requeridas por el cliente solamente controlan un tipo de dispositivo en específico. En el caso del DGM v2.9.4 se determinó tomarlo como base para el desarrollo del sistema.
- El enfoque ágil que brinda la metodología XP, el uso de las tecnologías y herramientas libres, permitieron realizar el diseño del SCCD en correspondencia con las especificaciones del cliente.
- La selección del estándar de codificación permitió la implementación del sistema garantizando el cumplimento de normas y reglas de implementación.
- La realización de las pruebas de aceptación y camino básico arrojaron resultados satisfactorios, permitiendo obtener una versión final estable del sistema.

Recomendaciones

El autor propone las siguientes recomendaciones:

- A la dirección del MININT valore integrar el Sistema para el Control Centralizado de los Dispositivos (SCCD) en su versión 1.0 con el Sistema Único de Identidad Nacional (SUIN), en aras de mantener un mayor control de los dispositivos que conforman la red.
- A la dirección del Centro de Identificación y Seguridad Digital (CISED) valore la conveniencia de agregar funcionalidades de captación biométrica (huellas dactilares y firma digital) al Servicio para el manejo de dispositivos.

Referencias

1. **Tinoco, Richard Dios.** Monografías.com. *Sistemas.* [En línea] 2011. http://www.monografias.com/trabajos87/sistemas-general/sistemas-general.shtml.

2. **Española, Real Academia.** Real Academia Española. [En línea] 2016. dle.rae.es.

3. **Herrera, Sujey Anahí Díaz.** *Hardware de una Computadora.* Universidad Autónoma del Estado de Hidalgo, México : s.n., 2012.

4. **W3c.es.** *Guía Breve de Servicios Web.* [En línea] 2016. http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb.

5. **Quin, Liam.** Extensible Markup Language (XML). [En línea] 2016. https://www.w3.org/XML/.

6. **Don Box, David Ehnebuske,Gopal Kakivaya,Andrew Layman,Noah Mendelsohn,Henrik Frystyk Nielsen,Satish Thatte,Dave Winer.** Simple Object Access Protocol (SOAP) 1.1. [En línea] 2016. https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

7. **Chase, Nicholas.** IBM developerWorks. *Comprender los servicios web, Parte 2: Web Services Description Language (WSDL).* [En línea] 2011. http://www.ibm.com/developerworks/ssa/webservices/tutorials/ws-understand-web-services2/.

8. **Rodriguez, Alex.** IBM developersWorks. *RESTful Web services: The basics.* [En línea] 2015. https://www.ibm.com/developerworks/library/ws-restful/.

9. **Ayala, Miguel Angel.** Diccionario de la Informática. [En línea] 2012. http://www.cdlibre.org/clase/03047m/amaya/Miguel_Angel_Ayala/diccionario/diccionario.html.

10. **Corporations, GNU.** ¿Qué es GNU/Linux? [En línea] 2015. http://www.grulic.org.ar/~mdione/www-grulic/trunk/linux.html.

11. **nddPrint.** [En línea] 2013. http://www.nddprint.com.br/360/ES/solucoes.php?id=1#.

12. **CZ Solution.** [En línea] 2012. http://www.czsolution.com/spanish/control-de-impresiones.htm.

13. **OSC Inventory NG.** [En línea] 2014. http://www.ocsinventory-ng.org/en/.

14. **Campos Rodriguez, Sandy y Rodríguez Sánchez, Hector Luis.** *Sistema para la interacción y control centralizado de dispositivos en aplicaciones web.* La Habana : s.n., 2012.

15. **Samira. Scribd.** *Metodologias de desarrollo de software.* [En línea] 2014. http://www.scribd.com//doc/2050925/metodologias-de-desarrollo-software..

16. **Penadés, Master. Carmen. CyTA.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* [En línea] 2014. http://www.cyta.com.ar/ta0502/v5n2a1.htm.

17. **Proyectos ágiles.org.** [En línea] 2015. https://proyectosagiles.org/desarrollo-iterativo-incremental/.

18. **R.Pressman.** *Ingeniería de SoftwareUn enfoque Práctico 7ma Edición.* University ofConnecticut : McGraw-Hill Interamericana Editores, S.A., 2010.

Referencias Bibliográficas

19. **Carlos Cortes, Vannesa Molina, Liseth Paternina, Oscar Vargas.** SlideShare. *metodologias agiles xp.* [En línea] 2013. http://es.slideshare.net/LisPater1/metodologias-agiles-xp.

20. **Avila, Kilmer Hernández.** Sistema para la Gestión de la Información del Control de la Venta de Pasajes en Divisa por Viazul. La Habana : s.n., 2015.

21. **UML, Página oficial de.** Unified Modeling Language™ (UML®) Resource Page. [En línea] 2015. http://www.uml.org/.

22. **Netbeans, Página oficial de.** Netbeans IDE-Overview. [En línea] 2015. https://netbeans.org/features/.

23. **NetBeans.** *NetBeans IDE.* [En línea] 2014. https://netbeans.org/community/releases/80/.

24. **Computación Aplicada al Desarrollo.** *Historia de lenguaje java.* [En línea] 2015. http://www.cad.com.mx/historia_del_lenguaje_java.htm.

25. **IBM.** *Introducción a programación Java.* [En línea] 2016. http://www.ibm.com/developerworks/ssa/java/newto/.

26. **JAVA.** *Que es Java. Características de Java como Lenguaje de programación.* [En línea] 2015. http://www.infor.uva.es/~jmrr/tgp/java/JAVA.html.

27. **Fernández, Adelaida Ramírez.** CLASIFICACIONES DE TIPOS DE REQUISITOS PARA LA MEJORA DEL PROCESO DE DESARROLLO DEL SOFTWARE. 2012.

28. **Sommerville, Ian.** *Ingienería del Software 9na Edición.* 2014.

29. **RAE.** Real Academia Española. *Real Academia Española.* [En línea] 2016. http://dle.rae.es/?id=W6xh4wt.

30. **Fuentes, María del Carme Gómez.** ConocimientosWeb.net. *Análisis de Requisitos.* [En línea] 2014. http://www.conocimientosweb.net/dcmt/ficha25180.html.

31. **Can, Carolina Novelo.** *Reporte de Instalación de Apache.* 2010.

32. **Escobar, Carlos Javier Pérez.** Qué significa CMMI . *Categorias de requisitos no funcionales .* [En línea] 2013. http://asprotech.blogspot.com/2013/09/categorias-de-requisitos-no-funcionales.html.

33. **Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas.** Manifiesto ágil. *Manifiesto por el Desarrollo Ágil de Software.* 2001.

34. **Medina, María.** Ingeniería de Software. [En línea] 2012. http://isittla12.blogspot.com/2012/11/unidad-3planificacion-del-proyecto-de.html.

35. **Joskowicz, José.** Reglas y Prácticas en eXtreme Programming. *Reglas y Prácticas en eXtreme Programming.* Universidad de Vigo, España : s.n., 2008.

36. **Entorno Virtual de Aprendizaje.** *EVA.* [En línea] 2015. http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodologias/Metodologias_agiles_en_el_desarrollo_de_software.pdf..

Referencias Bibliográficas

37. **Ganbetadev.** *Metodologías-de-programacion: patrones-de-diseno-que-son-y-por-que-debes-usarlos.* [En línea] 2016. http://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-que-son-y-por-que-debes-usarlos.

38. **Servicios, Líderes del grupo de Tecnología SOA de Accenture.** Arquitectura Orientada a. *Centro de Alto Rendimiento de Accenture (CAR).* 2008.

39. **Microsoft.** *Revisiones de código y estándares de codificación.* [En línea] 2014. https://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

40. **Idalgo, Eduardo Sánchez.** About.com. *Interfaz de Usuario.* [En línea] 2016. http://computadorasmac.about.com/od/nuevos-usuarios-mac/g/Interfaz-De-Usuario.htm.

41. **Gil, Luis Alexander Aldazabal.** Code2Read.com. *code metrics – Complejidad Ciclomática ¿Qué tan complejo es tu código?* [En línea] 2015. https://code2read.com/2015/03/25/code-metrics-complejidad-ciclomatica/.

42. **Shalloway, Alan y Trott, James.** *Desing Patterns Explained.* s.l. : Addison Wesley, 2004. ISBN: 9780321247148.

43. **Ayud, Prof. Asistente Noel Pérez.** Monografías.com. [En línea] 2016. http://www.monografias.com/usuario/perfiles/prof_instructor_noel_perez_ayup/datos.

Glosario de Términos

A	AJAX	*Asynchronous JavaScript And XML* (*JavaScript* asíncrono y XML): Es una técnica de desarrollo *web* para crear aplicaciones interactivas.
	AMD	*Advanced Micro Devices* microprocesadores desarrollados por una empresa estadounidense del mismo nombre.
	API	*Application Programming Interface* (Interfaz de Programación de Aplicaciones): Es una interfaz de comunicación entre componentes de software. Uno de los principales consiste en proporcionar un conjunto de funciones de uso general además de representar un método para conseguir abstracción en la programación.
C	C	Es un lenguaje orientado a la implementación de Sistemas Operativos.
	C#	Es un lenguaje de programación que se ha diseñado para compilar diversas aplicaciones que se ejecutan en *.NET Framework.*
	C++	Es un lenguaje imperativo orientado a objetos derivado de C.
D	DELETE	Permite eliminar un objeto (instancia de identidad).
F	FTP	*File Transfer Protocol*, (Protocolo de Transferencia de Archivos): en informática, es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (*Transmission Control Protocol*), basado en la arquitectura cliente-servidor.
G	GET	Permite obtener un valor. Puede ser un listado de objetos.
	GNU	GNU es un acrónimo recursivo que significa GNU No es Unix (*GNU is Not Unix*).
	GRASP	Son patrones generales de software para asignación de responsabilidades, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.
	GUI	*Graphical User Interface* (Interfaz Gráfica de Usuario): Conjunto de componentes gráficos que posibilitan la interacción entre el usuario y la aplicación.
H	HTML	*HyperText Markup Language* (Lenguaje de Marcas de Hipertexto): Utiliza marcas para describir la forma en la que deberían aparecer el texto y los gráficos en un Navegador *web* que, a su vez, están preparados para leer esas marcas y mostrar la información en un formato estándar.

	HTTP	*Hypertext Transfer Protocol* (Protocolo de transferencia de hipertexto): es el protocolo usado en cada transacción de la *World Wide Web*, o WWW.
	i386	Intel 80386: es un microprocesador con arquitectura x86 mejor conocido como P3 por ser el prototipo de la tercera generación x86.
I	i486	Intel 80486: son una familia de microprocesadores de 32 bits con arquitectura x86 diseñados y fabricados por *Intel Corporation.*
	IDE	*Integrated Development Environment* (Entorno de Desarrollo Integrado o Entorno de Desarrollo Interactivo): Es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de *software*.
	IP	*Internet Protocol* (Protocolo de Internet): Es un protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.
M	MAC	Dirección física de los ordenadores.
O	OPP	*Object Oriented Programming* (Programación orientada a objeto): Forma especial de programación que permite representar de un modo mucho más conveniente al mundo real que en otros tipos de programación.
P	PDF	*Portable Document Format* (Formato de Documento Portátil): Es un formato de almacenamiento para documentos digitales independiente de plataformas de software o hardware.
	PHP	*Personal Home Page* (Página Personal de Inicio): Lenguaje de programación, interpretado, diseñado originalmente para la creación de Páginas *web* dinámicas.
	POST	Permite guardar un nuevo objeto (instancia de identidad) en la aplicación.
	PUT	Permite actualizar un objeto.
R	RAM	*Random Access Memory* (Memoria de Acceso Aleatorio): Es donde el computador guarda los datos que está utilizando en el momento presente.
	RBAC	Control de acceso basado en roles.
T	TCP/IP	Protocolo de control de transmisión/Protocolo de Internet. representa todas las reglas de comunicación para Internet y se basa en la noción de dirección IP, es decir, en la idea de brindar una dirección IP a cada equipo de la red para poder enrutar paquetes de datos.

U	UML	*Unified Modeling Language*: Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.
	USB	*Universal Serial Bus:* Periférico que permite conectar diferentes periféricos a una computadora.

Tabla 15: Glosario de Términos. Fuente: Elaboración Propia.

Anexos

Anexo 1: Historias de usuario

Historia de Usuario	
Numero: HU_2	**Usuario:** Administrador
Nombre de la historia: Gestionar las cámaras conectadas (por *USB*) a las estaciones de trabajo.	
Prioridad en negocio: Muy Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3	**Iteración asignada:** 1
Responsable: Joseph Michael Cordero Korolev	
Descripción: La HU se inicia cuando el administrador accede al *framework JS* utilizando el navegador *web*. En el mismo puede gestionar la cámara que esté conectada a la estación de trabajo utilizando los siguientes botones: 1. Capturar: Permite capturar la imagen en tiempo real que esté siendo visualizada en ese momento. 2. Guardar: Guardar la imagen capturada.	
Observaciones: 1. Deben estar disponibles los servicios *web*.	

Tabla 16: HU_2. Gestionar las cámaras conectadas (por USB) a las estaciones de trabajo. Fuente: Elaboración Propia.

Historia de Usuario	
Numero: HU_5	**Usuario:** Administrador
Nombre de la historia: Actualizar el Servicio	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Alto
Puntos estimados: 2	**Iteración asignada:** 3
Responsable: Joseph Michael Cordero Korolev	

Descripción: 1. Se selecciona la opción “Actualizar” en el Panel de Control del SCCD. 2. Conectar el actualizador al servicio de actualización. 3. Descargar la información necesaria para llevar a cabo el proceso de actualización. 4. Detener todos los servicios *web*. 5. Descargar todos los ficheros necesarios para la actualización. 6. Reemplazar todos los ficheros antiguos. Iniciar todos los servicios *web* ya actualizados.
Observaciones: 1. El servicio de actualización y la aplicación de actualización deben estar en ejecución y correctamente configuradas.

Tabla 17: HU_5. Actualizar el Servicio. Fuente: Elaboración Propia.

Anexo 2: Tarjetas CRC

webcamUSB	
Responsabilidades	**Colaboradores**
Es la que permite el control de las cámaras USB que se encuentren conectadas a las estaciones de trabajo.	----------

Tabla 18: Tarjeta CRC_3. webcamUSB. Fuente: Elaboración Propia.

Actualizar	
Responsabilidades	**Colaboradores**
Se encarga de actualizar el servicio SCCD.	----------

Tabla 19: Tarjeta CRC_5. Actualizar. Fuente: Elaboración Propia.

Anexo 3: Casos de prueba de aceptación

Caso de Prueba de Aceptación	
Código: CP2_HU2	**HU_ 2:** Gestionar las Cámaras Conectadas (por USB) a las Estaciones de Trabajo.
Responsable de la prueba: Joseph Michael Cordero Korolev.	
Descripción: Interactuar con un dispositivo remotamente.	
Condiciones de ejecución: 1. La estación de trabajo estar conectada a la red. 2. La cámara debe estar conectada a la estación de trabajo. 3. El servicio debe estar iniciado.	
Entrada / Pasos de ejecución:	

1. Acceder el servicio. 2. Comprobar si la cámara está conectada a la estación de trabajo. 3. Si está conectada. 3.1. Capturar la imagen. 4. Si no está conectada. 4.1 Mostrar un mensaje de error.
Resultado esperado: Se obtuvo la imagen.
Evaluación de la prueba: Satisfactoria

Tabla 20: Caso de prueba de aceptación correspondiente a la HU_2. Fuente: Elaboración Propia.

Caso de Prueba de Aceptación	
Código: CP5_HU5	**HU_ 5:** Actualizar el Servicio.
Responsable de la prueba: Joseph Michael Cordero Korolev.	
Descripción: Mantener el servicio actualizado.	
Condiciones de ejecución: El servicio de actualización y la aplicación de actualización deben estar en ejecución y correctamente configuradas.	
Entrada / Pasos de ejecución: 1. Se selecciona la opción “Actualizar” en el Panel de Control del SCCD. 2. Conectar el actualizador al servicio de actualización. 3. Descargar la información necesaria para llevar a cabo el proceso de actualización. 4. Detener todos los servicios *web*. 5. Descargar todos los ficheros necesarios para la actualización. **6.** Reemplazar todos los ficheros antiguos. Iniciar todos los servicios *web* ya actualizados.	
Resultado esperado: El Servicio para el manejo de dispositivos con todos los servicios *web* se encuentran actualizados.	
Evaluación de la prueba: Satisfactoria.	

Tabla 21: Caso de prueba de aceptación correspondiente a la HU_5. Fuente: Elaboración Propia.

Anexo 4: Interfaz de usuario

Ilustración 17: Interfaz de usuario del SCCD. Servicio de webcam iniciado. Fuente: Elaboración Propia.

Ilustración 18: Mensaje informativo de activación del servicio webcam. Fuente: Elaboración Propia.

Ilustración 19: Mensaje informativo de detención del servicio webcam. Fuente: Elaboración Propia.

Ilustración 20: Interfaz de usuario del SCCD. Servicio de impresión iniciado. Fuente: Elaboración Propia.

Ilustración 21: Mensaje informativo de activación del servicio de impresión. Fuente: Elaboración Propia.

Ilustración 22: Mensaje informativo de detención del servicio de impresión. Fuente: Elaboración Propia.