Universidad de las Ciencias Informáticas

Título:

Sistema de Gestión y Control de los Activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.

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

Autores:

Angel Antonio Guevara Hernández

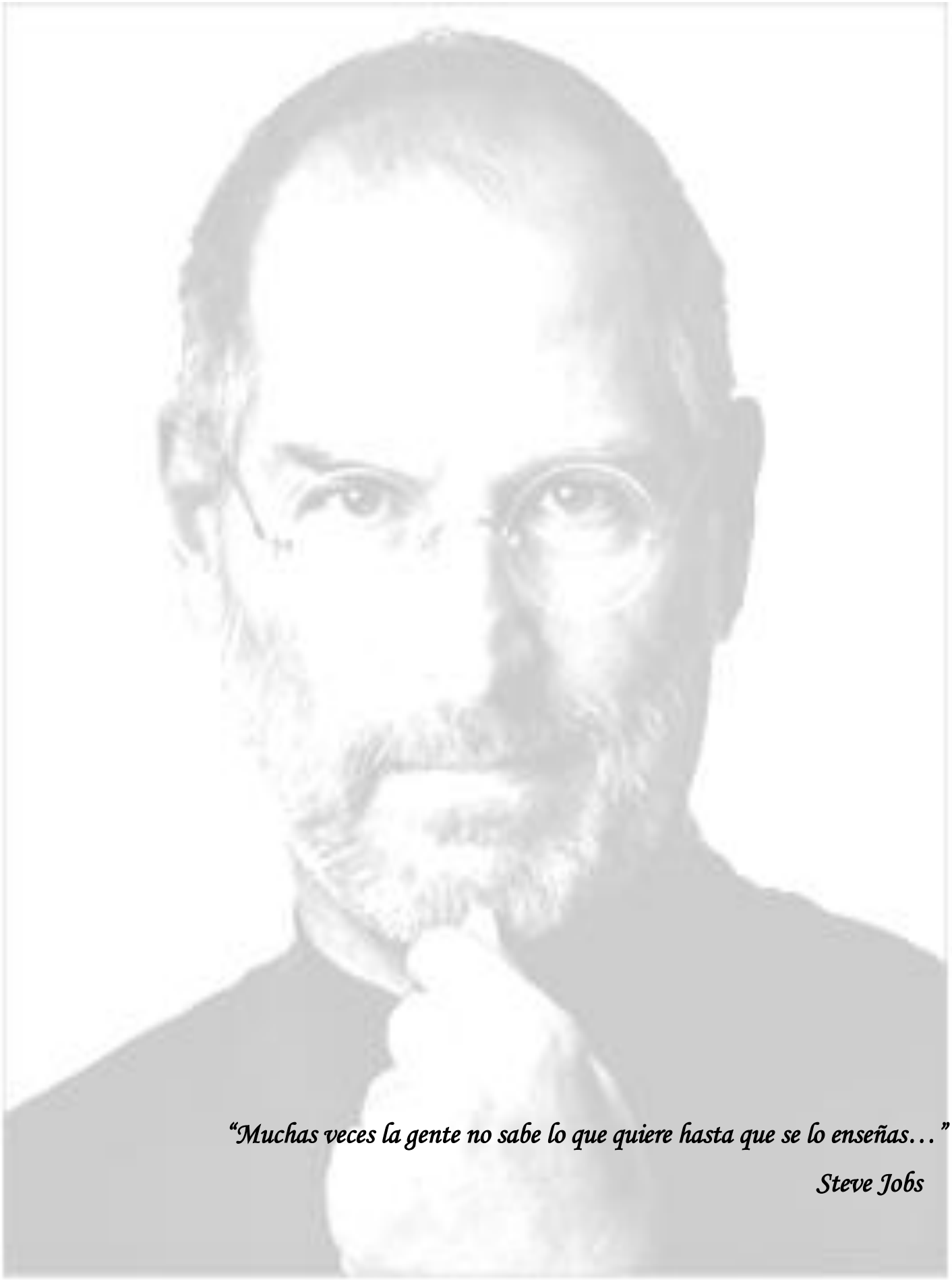
Julio Antonio Lee Fonseca

Tutor: Ing. Roberto Llerena Villar

Co-tutor: Ing. Iran Roberto Rodríguez Moreno

Ciudad de la Habana, Cuba

Junio, 2010



"Muchas veces la gente no sabe lo que quiere hasta que se lo enseñas…"

Steve Jobs

Declaración de autoría

Declaramos que somos los únicos autores del trabajo titulado: “Sistema de Gestión y Control de los Activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas”, y otorgamos 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 2010.

Angel Antonio Guevara Hernández

_____

Firma del Autor

Julio Antonio Lee Fonseca

_____

Firma del Autor

Ing. Roberto Llerena Villar

_____

Firma del Tutor

Ing. Iran Roberto Rodríguez Moreno

_____

Firma del Co-tutor

Datos de contacto

Angel Antonio Guevara Hernández
Correo: aaguevara@estudiantes.uci.cu.
Ciudad de La Habana, Cuba.

Julio Antonio Lee Fonseca
Correo: jalee@estudiantes.uci.cu.
Ciudad de La Habana, Cuba.

Ing. Roberto Llerena Villar
Ingeniero en Ciencias Informáticas.
Correo: rllerena@uci.cu.
Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Ing. Iran Roberto Rodríguez Moreno
Ingeniero en Ciencias Informáticas.
Correo: irrodriguez@uci.cu.
Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Agradecimientos

Quiero agradecer primeramente a mis padres por todo el apoyo que siempre me han dado y por confiar ciegamente en mí, espero no defraudarlos nunca.

A Adis y a Novoa por acogerme como su hijo varón y por cuidar de mis padres.

A mi novia Yadira por ayudarme tanto en el desarrollo de este trabajo y por guiarme por el buen camino.

Agradecer a Adianez, Ibrael, Imer y Mario por su gran contribución en este trabajo.

A todos mis amigos.

A mis tutores por su excelente tutoría, al tribunal y a todas las personas que de una forma u otra colaboraron con el desarrollo de esta tesis.

Angel

A mis padres, a mis abuelos, a mis hermanos, en general a toda mi familia, por el apoyo que me han dado en toda mi carrera estudiantil.

A mis tutores por todo el apoyo ofrecido en la tutoría de la tesis, así como al tribunal y a todas las personas que han colaborado en el desarrollo de la misma.

A mis amigos por estar siempre en el momento que los he necesitado.

Julio

Dedicatoria

A toda mi familia, que es la mejor del mundo. Especialmente a mis padres y a Yadira, espero algún día poder pagarles todo lo que han hecho por mí.

Angel

A mis padres, abuelos y hermanos y a mi compañero de tesis.

Julio

Resumen

En la actualidad, las Telecomunicaciones constituyen uno de los sectores más importantes para cualquier país ya que contribuye al desarrollo económico, social y mejora la calidad de vida de la población. En la Universidad de las Ciencias Informáticas (UCI) se trabaja con un gran número de activos de Telecomunicaciones, donde el manejo de estos es complejo debido a la dinámica de esta universidad. Además, no existe un sistema especializado para su gestión y control, por lo que se hace necesario el desarrollo de un sistema que gestione estos activos garantizando un mejor control en el manejo de los mismos. Por lo anteriormente mencionado se desarrolla un sistema capaz de gestionar y controlar el flujo de actividades que se realizan con los activos de Telecomunicaciones en la universidad. Para ello, se estudian varios sistemas de gestión de activos, de los cuales se toman importantes funcionalidades para la implementación de la propuesta de solución. Además se estudia la metodología, así como las herramientas, lenguajes y tecnologías, propuestas por el cliente, con los que se desarrolla el Sistema de Gestión de Activos de Telecomunicaciones (SGAT), en la Universidad de las Ciencias Informáticas, para lograr un producto de alta calidad y satisfacer todas las exigencias del cliente.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

La especie humana es de carácter social, necesita de la comunicación; pues de otra manera viviríamos completamente aislados. Así, desde los inicios de la especie, esta fue evolucionando hasta llegar a la más sofisticada tecnología, para lograr acercar espacios y tener mayor velocidad en el proceso de comunicación.

Unas de las primeras manifestaciones en la comunicación de la especie humana fueron: la voz, las señales de humo y sus dibujos pictóricos; posteriormente al evolucionar, se fueron creando otros como el telégrafo, el teléfono cableado y la radio, pero todos estos avances tecnológicos y necesidades de comunicar y transmitir datos fueron tomando cada vez más auge alcanzando un gran desarrollo de la tecnología a nivel mundial, conduciéndonos rápidamente hacia la sociedad del conocimiento del futuro, en la que prácticamente cualquier clase de información y servicios estará disponible en cualquier parte del mundo y nuestra capacidad de comunicación con todo el planeta será inmensa.

Todo este impulso de las comunicaciones dio lugar a una nueva técnica; que consiste en la transmisión, emisión o recepción de signos, señales, datos, imágenes, voz, sonidos o información de cualquier naturaleza que se efectúa a través de cables, radioelectricidad, medios ópticos, físicos u otros sistemas electromagnéticos, más conocida como Telecomunicaciones.

Cuba no es ajena a estos avances, más aún, si tomamos en cuenta los progresos científicos que en los últimos tiempos se han venido alcanzando en materia de Tecnologías de la Información y las Comunicaciones (TIC). Para ello se ha trabajado en 3 grandes proyectos como son:

- Informatización de la Salud Pública.
- Informatización de los Joven Club de Computación y Electrónica.
- Informatización de la Educación.

Gracias a la revolución llevada a cabo en el sector educacional surge en el año 2002 la Universidad de las Ciencias Informáticas (UCI), principal pilar de la producción de software en el país.

El trabajo con la tecnología en la Universidad de las Ciencias Informáticas, es complejo y no existe un sistema especializado en la gestión y control digital de los activos de Telecomunicaciones que hacen posible el enlace de la información; dígase como activos de Telecomunicaciones: conmutador (en el área de redes), plantas, beeper y pin (en el área de telefonía móvil). Hasta el momento el control de estos activos se realiza a mano, trayendo consigo:

- Desactualización constante de información sobre los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.
- Pérdida de información.
- Duplicidad de la información.
- Demora en la prestación de servicios para la reparación de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.
- Difícil y tediosa la búsqueda de propiedades y características de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.

Por las insuficiencias antes planteadas surge el siguiente **problema científico**: ¿Cómo mejorar los procesos de gestión y control de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas referidos a conmutador, planta, beeper y pin?

Por tanto el presente trabajo centra su **objeto de estudio** en los procesos de gestión y control de los activos de Telecomunicaciones.

Campo de acción: gestión y control de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas referidos a conmutadores, plantas, beeper y pin.

Hipótesis: si se desarrolla un sistema que gestione los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas se garantizará un mejor control en el manejo de estos.

Variables

Variable independiente: sistema de gestión de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.

Variable dependiente: control en el manejo de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.

Objetivo general: desarrollar un sistema para gestionar y controlar los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas. Obteniéndose los siguientes **objetivos específicos**:

- Estudiar el marco teórico de la investigación sobre la gestión y control de los activos de Telecomunicaciones.
- Modelar los procesos relacionados con los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.
- Diseñar e Implementar el sistema para la gestión y control de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas referidos a conmutadores, plantas, beeper y pin.
- Validar el sistema para la gestión y control de los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas referidos a conmutadores, plantas, beeper y pin.

Posibles resultados: con este trabajo se obtendrá un sistema que gestionará y controlará los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas, facilitando el trabajo al personal de redes y comunicaciones en la universidad.

Tareas de la investigación científica:

1. Realización de entrevistas al personal de la Dirección de Gestión Tecnológica de la Universidad de las Ciencias Informáticas para describir los procesos de gestión y control de los activos de Telecomunicaciones en la universidad.

2. Realización de un estudio de los Software de Gestión relacionados con la gestión y control de activos de Telecomunicaciones a nivel mundial, nacional y en la Universidad de las Ciencias Informáticas.

3. Realización de un estudio de las herramientas, metodologías y tecnologías a utilizar en la propuesta de solución.

4. Descripción de los requerimientos funcionales y no funcionales del software a desarrollar.

5. Definición la arquitectura, patrones de diseño, modelo global del sistema y lista de funcionalidades.

6. Planeación de las funcionalidades del sistema.

7. Realización del diseño e implementación del sistema de acuerdo a los requerimientos y procesos descritos.

8. Implementación de los casos de prueba.

9. Validación del sistema implementado para comprobar que el mismo se ajusta a las necesidades del cliente realizando las pruebas.

10. Presentación al cliente del sistema para verificar que se ajusta a sus necesidades.

11. Descripción de los resultados obtenidos de la investigación.

Para garantizar un desarrollo correcto de la investigación es necesario el empleo de **Métodos Científicos.**

Métodos teóricos

Analítico-Sintético: se consulta una amplia bibliografía con el objetivo de conocer el estado del arte de los sistemas relacionados con la investigación en el mundo y se obtienen de ellos ideas que ayudan en la realización del sistema.

Histórico-Lógico: se realiza un estudio histórico-lógico de la gestión y control de los activos de Telecomunicaciones. Además se estudian soluciones existentes relacionadas con el problema a resolver.

Métodos empíricos:

Entrevista: se realizan entrevistas al personal de redes y Telecomunicaciones de la universidad para conocer las dificultades existentes que dan pie al problema científico.

Experimental: para comprobar si la solución propuesta es exitosa se realizan las pruebas pertinentes para comprobar su calidad.

Modelación: se utiliza durante la elaboración del Sistema de Gestión y Control de los Activos de Telecomunicaciones en la UCI con el objetivo de que el cliente comprenda de una manera más fácil cómo quedará el sistema.

Estructura del trabajo

Capítulo I: Fundamentación teórica.

En este capítulo se dan a conocer las cuestiones teóricas necesarias para la comprensión del trabajo lo cual incluye un estudio del estado del arte sobre Software de Gestión a nivel

mundial, nacional y en la UCI. Se realiza un análisis de la metodología y herramientas necesarias para el desarrollo de la solución; así como una justificación de su uso.

Capítulo II: Características y diseño del sistema.

En este capítulo se realiza una caracterización del sistema definiendo sus principales funcionalidades, además se realiza todo el diseño de la aplicación con los artefactos definidos en esta fase. Se determina además la arquitectura del sistema y los patrones de diseño a utilizar en la implementación.

Capítulo III: Implementación y prueba.

Se realiza la implementación del sistema con sus respectivos diagramas de componentes y de despliegue, posteriormente se realizan las pruebas necesarias para el chequeo de la aplicación.

CAPÍTULO I: Fundamentación teórica

1.1 Introducción

En el siguiente capítulo se realiza un estudio del estado del arte de los diferentes sistemas de gestión de activos a nivel internacional y en el país, que puedan dar una solución a la problemática planteada en la investigación. Además, se hace un estudio de las herramientas, tecnologías, metodología y lenguajes sugeridos por el cliente, para dar solución a dicha problemática, especificándose sus principales características y las ventajas de su utilización.

1.2 Sistemas de gestión

Sistema: según la Real Academia, organismo responsable de elaborar las reglas normativas del español, plasmadas en el diccionario, la gramática y la ortografía: proviene del latín (*systema*) y significa "conjunto de reglas o principios sobre una materia racionalmente enlazados entre sí o conjunto de cosas que relacionadas entre sí ordenadamente contribuyen a determinado objeto". (1)

Gestión: según la Real Academia, el término Gestión es la "acción y efecto de gestionar, donde gestionar es hacer diligencias conducentes al logro de un negocio o de un deseo cualquiera". (1)

Sistemas de Gestión: un sistema de gestión es una estructura probada para la gestión y mejora continua de las políticas, los procedimientos y procesos de la organización. (2)

Desde el surgimiento de las primeras organizaciones, empresas e instituciones de diversas índoles siempre ha sido una necesidad un sistema de gestión el cual puede ser utilizado con diversos propósitos según la esfera donde se desee implantar dicho sistema. Solo por citar algunos ejemplos, un sistema de gestión puede gestionar distintos tipos de datos como información referente a escuelas, empresas, universidades y centros investigativos. Todos estos sistemas de gestión tienen algo en particular ya que están dirigidos a distintas áreas, es por esto que existen clasificaciones para estos sistemas de gestión según su propósito, como por ejemplo:

Sistema de Gestión de Contenido: (CMS siglas por el nombre en inglés *Content Management System*) permite la creación y administración de contenidos principalmente en páginas web.

Consiste en una interfaz que permite manejar de manera independiente el contenido por una parte y el diseño por otra. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido nuevamente, además de permitir fácil y controladamente la publicación en el sitio por varios editores.

Un Sistema de Gestión de Contenido permite que usuarios sin conocimientos técnicos ni de diseño de páginas web puedan actualizar sus sitios, añadiendo secciones, noticias, páginas o productos con relativa facilidad. (3)

Sistema de Gestión Comercial: es un sistema de registración y emisión de comprobantes que abarca las funciones tradicionales de cualquier empresa, facturación de ventas y compras, control de inventarios, control de cuentas corrientes, registración contable y cálculo de impuestos, control de disponibilidades financieras y registración de ingresos y egresos de dinero, como lo es ERP (*Enterprise Resource Planning*- Planificación de Recursos Empresariales), que además de la gestión tradicional, brinda la facilidad de presupuestar, sacar estadísticas, generar proyecciones en base a históricos, lo que permite optimizar el rendimiento comercial. (4)

Sistema de Gestión de Calidad: realizado con el objetivo de orientar las actividades de una empresa para obtener y mantener el nivel de calidad del producto o el servicio, de acuerdo con las necesidades del cliente, proporcionando actividades coordinadas para dirigir y controlar una organización en lo relativo a la calidad. (5)

1.3 Sistemas automatizados existentes

Los sistemas de gestión, ya sean de uno u otro tipo, contribuyen a gestionar los riesgos sociales, medioambientales y financieros, mejorar la efectividad operativa, reducir costos y lograr mejoras continuas.

Debido a que el objetivo principal de este trabajo es el desarrollo de un sistema de gestión y control de activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas surge la necesidad de investigar sobre sistemas de gestión de activos en el ámbito internacional, nacional y en nuestra universidad.

1.3.1 Soluciones Internacionales

Proga: es un sistema de gestión eficiente de activos públicos que maximiza el retorno económico y social de los activos del estado colombiano, genera mejoría en la eficiencia y la eficacia de la gestión pública, dado que los recursos físicos de las entidades estatales son proporcionales a sus necesidades y se eliminan las erogaciones destinadas al mantenimiento de activos ociosos. Este sistema se enfoca en los activos de inmuebles de las entidades. (6)

Este sistema realiza una gestión detallada de los inmuebles, lo cual resulta actractivo e importante ya que puede ser utilizado por personas de poco conocimiento del tema. Por lo que se tendrá en cuenta la manera en que se maneja la información durante la realización del sistema, con el objetivo de que aquellas personas que interactúen con el mismo se familiaricen de una manera rápida con la aplicación.

DSpace: es un sistema de gestión de activos digitales, que fue elaborado conjuntamente por el MIT y HP Labs. Está diseñado para capturar, almacenar, indexar, preservar y redistribuir enormes cantidades de datos digitales y se ejecuta en sistemas Unix, Linux y Microsoft Windows. Este sistema dirige su gestión a fotos digitales, documentos word, excel y música (Colección Digital). (7)

Es importante señalar que este sistema permite realizar búsquedas avanzadas sobre los activos que maneja, lo cual se tiene en cuenta en la propuesta de solución cuando se realizan reportes dinámicos de los activos manejados por la aplicación desarrollada.

1.3.2 Soluciones Nacionales

El VERSAT-Sarasola: primer sistema de contabilidad cubano certificado, en cuya evaluación participaron el Ministerio de Finanzas y Precios, consultorías internacionales y el organismo encargado de la seguridad informática, es un paquete integrado para la gestión económica financiera que permite enviar información eficaz, de forma inmediata, desde lugares apartados, a la vez que ofrece mayor organización, control y disciplina en cada gestión. Fue creado por el villaclareño Miguel Cabrera González y se aplicó por primera vez en el central azucarero George Washington, en 2001. Fue el primer sistema cubano que logró certificarse en el país y está estructurado con los elementos requeridos por la contabilidad de este, con lo

cual se ha ganado la aceptación en el mercado y ha logrado diseminarse por un gran número de entidades nacionales. (8)

RODAS XXI Versión 3.0: sistema multiempresa y multiusuario creado por CITMATEL para la automatización de la gestión empresarial. Contiene diferentes módulos que pueden usarse integrados o independientes dentro de los cuales está el de activos fijos tangibles. El módulo de activos Fijos de RODAS XXI permite:

- Tener un control detallado de los activos fijos de su entidad, realizando en el mismo momento que se registra un movimiento, su contabilización.
- Realizar operaciones de activos fijos en el momento que se desee, generando el documento asociado al movimiento de que se trate de forma automática, previa configuración del sistema para ello.
- Permite el control por separado de los activos fijos que se encuentran en almacén de los que se encuentran en explotación.
- Este módulo da la posibilidad de realizar la depreciación de forma automática
- El comprobante de depreciación se genera, al igual que con los movimientos de activos fijos, de forma automática.
- Permite obtener el submayor de activos fijos, listados y localización de los medios de transporte de la entidad, la depreciación mensual y acumulada de uno o de los activos fijos que desee, el acta de responsabilidad material de cada una de las áreas.
- Visualiza información correspondiente a períodos anteriores, tan solo con cambiar de período contable a períodos anteriores ya cerrados, aunque en dichos períodos no podrá realizar ninguna operación. (9)

Varias de estas funcionalidades que permite la aplicación RODAS se tienen en cuenta en la realización del sistema, por la similitud con el negocio que se maneja en la dirección de Telecomunicaciones.

CEDRUX: es un sistema de gestión empresarial que se está desarrollando actualmente en la UCI. Este sistema está compuesto por diversos módulos de gestión especializados en diversas áreas. Este incluye un subsistema para el control de los activos fijos tangibles (AFT), que pretende servir de apoyo a la toma de decisiones en las entidades. El subsistema es capaz de gestionar activos de forma dinámica, tributa mediante operaciones al sistema

contable de la empresa y estandariza el tratamiento de los AFT a nivel nacional. Incluye además temas como la multimoneda, la agrupación de activos y la generación de documentos, prestaciones del sistema que responden a necesidades del país, actualmente no satisfechas de manera conjunta por los sistemas existentes. (10)

Este sistema posee una interfaz visual muy amigable lo que lo hace fácil de manejar. Además, posee un sistema de trazas que proporciona seguridad a la aplicación, las cuales son características que se persiguen en el sistema a desarrollar, por lo que se tienen en cuenta en la solución.

Valoración crítica de los sistemas existentes

Luego de un estudio realizado a las soluciones, se concluyó que los mismos no son factibles para la gestión de los activos de Telecomunicaciones de la universidad. Los sistemas internacionales utilizan tecnologías que no son accesibles para Cuba debido a las restricciones impuestas por Estados Unidos. Como otra de las desventajas que presentan estas soluciones es que han sido implementadas sobre herramientas propietarias provocando gastos muy elevados al país por conceptos de compra de licencias de software y mantenimiento. Además, algunos de los sistemas antes expuestos son aplicaciones de escritorio, lo que trae consigo que el usuario deba instalar la aplicación en cada estación de trabajo. La mayoría de estos sistemas se caracterizan por ser soluciones no adaptables a las necesidades del cliente. Vale aclarar que el sistema CEDRUX podría dar solución a la gestión de los activos de Telecomunicaciones de la UCI, pero el mismo posee la desventaja de que se encuentra actualmente en fase de desarrollo. Es importante señalar que son tomadas características y funcionalidades de estos sistemas para la implementación de la propuesta de solución.

1.4 ¿Por qué una aplicación web y no una aplicación de escritorio?

El sistema se desarrollará mediante una aplicación web porque las condiciones naturales del negocio las permiten y se pueden adaptar muy fácilmente a estos tipos de sistemas, en otras palabras el negocio es por naturaleza adaptable a un sistema web. Los programas de computación basados en la web son multiplataforma para el cliente, no hay necesidad de actualizaciones y son centralizados; a diferencia de las aplicaciones de escritorio, siempre

están al día e instantáneamente disponibles, no importa dónde esté el usuario o qué sistema operativo esté utilizando.

1.5 Metodologías de desarrollo

"*Desarrollar un buen software depende de un sinnúmero de actividades y etapas, donde el impacto de elegir la mejor metodología para un equipo en un determinado proyecto, es trascendental, para el éxito del producto".* (11)

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software. Las metodologías de desarrollo de software contribuyen a mejorar la calidad y a realizar un software en el tiempo esperado y con el coste estimado.

Estas van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, proponen qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla. (12)

Las tendencias presentes, luego del perfeccionamiento de los procesos del software durante años, han llevado a cabo dos corrientes significativas: los llamados métodos ligeros y métodos pesados. Aunque ambos están enfocados a beneficiar la labor de aquellas personas que intervienen en el proceso de desarrollo.

Los métodos ligeros o ágiles, proponen mejorar la calidad del software teniendo como premisa la comunicación inmediata y directa, mientras que los métodos pesados obtienen sus resultados a través de orden y documentación.

Las metodologías se definen por pasos a seguir para el cumplimiento de un objetivo. El objetivo dentro del desarrollo del software es producir un producto de alta calidad que cumpla con los requerimientos del cliente.

<u>Metodologías de desarrollo de software:</u>

XP (*eXtreme Programming*)

FDD (*Feature Driven Development*)

MSF (*Microsoft Solution Features*)

RUP (*Rational Unified Process*)

SCRUM

SXP(SCRUM+XP)

Crystal *Metodologies*

DSDM(*Dynamic Systems Development Method*)

1.6 ¿Por qué usar una metodología ágil?

Las metodologías ágiles de desarrollo están especialmente indicadas en proyectos con requisitos poco definidos o cambiantes. Estas metodologías se aplican bien en equipos pequeños que resuelven problemas concretos, lo que no está reñido con su aplicación en el desarrollo de grandes sistemas, ya que una correcta modularización de los mismos es fundamental para su exitosa implantación. Dividir el trabajo en módulos abordables minimiza los fallos y el coste. Las metodologías ágiles presentan diversas ventajas, entre las que podemos destacar:

- Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos breves de software funcional.
- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Importancia de la simplicidad, eliminando el trabajo innecesario.
- Atención continua a la excelencia técnica y al buen diseño.
- Mejora continua de los procesos y el equipo de desarrollo. (13)

1.6.1 FDD (Feature Driven Development)

FDD con sus siglas en inglés *Feature Driven Development* o Desarrollo Basado en Funcionalidades es un enfoque ágil para el desarrollo de sistemas. Dicho enfoque no hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Sin embargo, fue diseñado para trabajar con otras actividades de desarrollo de software y no requiere la utilización de ningún modelo de proceso específico. FDD está pensado para proyectos con tiempo de desarrollo relativamente cortos (menos de un año). Se basa en un proceso iterativo con iteraciones cortas (2 semanas) que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar. Además, hace énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del

avance del proyecto. Al contrario de otras metodologías, FDD afirma ser conveniente para el desarrollo de sistemas críticos.

Consiste en cinco fases secuenciales que son:

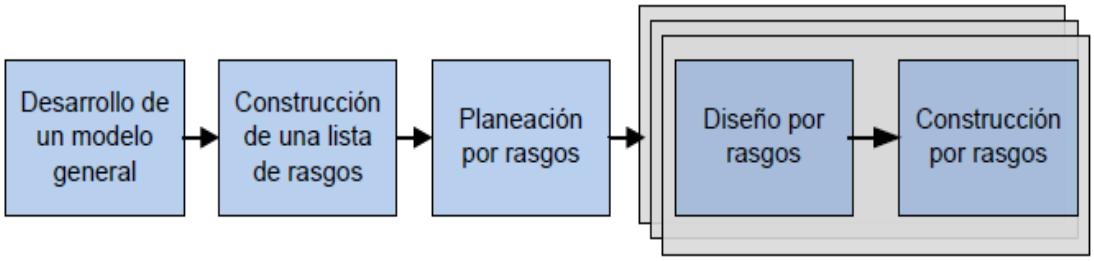


Figura 1. Fases de desarrollo de la metodología FDD

- **Desarrollo de un modelo general**: al iniciar esta fase, se tiene una idea del contexto y visión del sistema. Se presenta un ensayo del dominio (*walkthrough*) en el cual los miembros del equipo son informados a través de una descripción del sistema que se quiere construir. El dominio global es dividido en diferentes áreas y se realiza un ensayo del dominio detallado para cada una de las áreas del dominio. A partir de estas descripciones del contexto y visión del sistema se realiza un modelo de objetos para cada área de dominio y simultáneamente, se construye un modelo global del sistema a partir de los modelos por áreas. (14)
- **Construcción de una lista de Funcionalidades**: se identifican los rasgos que resumen las características y comportamiento del sistema que se desea construir. El resultado de esta fase es una lista de funcionalidades categorizada jerárquicamente. La lista de funcionalidades se compone por áreas temáticas, actividades del negocio que comprenden estas áreas temáticas y por las funcionalidades que representan los pasos para cumplimentar cada actividad del negocio. Estos rasgos son pequeñas funcionalidades útiles a los ojos del cliente y los rasgos que requieran de más de diez días se descomponen en otros más pequeños que se puedan cumplimentar en el tiempo máximo de dos semanas. (14)
- **Planeación por funcionalidades**: se incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se ponen en secuencia conforme a su prioridad y dependencia. Esta planificación permite establecer qué funcionalidades se incluyen en cada iteración de los dos últimos procesos que establece esta metodología. (14)

- **Diseño y construcción por funcionalidades**: este proceso se realiza de forma iterativa. En cada iteración se selecciona un pequeño conjunto de funcionalidades de la lista elaborada en la fase anterior. Se identifican las clases involucradas por cada funcionalidad y se diseñan e implementan estas clases a partir del conjunto de funcionalidades que están en la iteración. Se procede luego iterativamente hasta que se producen o se implementan todas las funcionalidades identificadas. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo está definido por los hitos: un ensayo del dominio del rasgo a desarrollar, diseño de las clases, inspección del diseño, codificación, pruebas unitarias, inspección de código y por último promoción de la funcionalidad que se construyó. (14)

En las iteraciones iniciales las tres primeras fases son las que ocupan más tiempo, pero a medida que avanza el proyecto entonces son las de diseño y construcción las que ocupan la mayoría del tiempo, quedando las tres primeras fases solo en un proceso de refinamiento.

Analizando lo anteriormente expuesto, el uso de la metodología ágil FDD brinda además las siguientes ventajas:

- En cuanto a la relación con el cliente, al principio de las fases se define el modelo global que produce un marco en el cual puede moverse el proyecto, no siendo necesario su cambio, a no ser casos especiales que lo requieran.
- Ayuda al equipo a producir resultados periódicos y tangibles.
- Hace énfasis en la obtención de resultados cada dos semanas.
- Asegura en gran parte la calidad del software entregado.
- Posibilita la conformación del equipo de trabajo y que estos creen su propio entorno de trabajo.
- Solamente es necesaria la creación de los documentos en caso que se requiera, la metodología no propone plantillas de trabajo, las que se deseen elaborar deben ser un documento corto y fácil de entender.

1.7 Lenguaje de modelado UML

El Lenguaje Unificado de Modelado (UML) (*Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Es el lenguaje de modelado de

sistemas de software más conocido y utilizado en la actualidad. Ofrece un estándar para describir un “plano” del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un “lenguaje” para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Los diagramas fundamentales que emplea UML son los siguientes:

Diagrama de casos de uso: especifica las funcionalidades y el comportamiento de un sistema, su interacción con los usuarios del mismo y otros sistemas.

Diagrama de comportamiento o interacción: muestra las interacciones entre objetos ocurridas en un escenario o parte del sistema.

Diagrama de clases: representa un conjunto de elementos del modelo que son estáticos. Estos aspectos estáticos modelan características del software como pueden ser su estructura interna y la representación que se hará de la información en la aplicación.

Diagrama de implementación: muestra los aspectos físicos del sistema e incluye la estructura del código fuente y la implementación. (15)

1.8 Herramientas CASE

Las herramientas CASE (Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y de dinero. Las herramientas CASE permiten organizar y manejar la información de un proyecto informático. Permite que los sistemas se tornen más flexibles, más comprensibles y además mejorar la comunicación entre los participantes.

Estas herramientas pueden servir de apoyo en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, compilación automática, documentación o detección de errores entre otras. (16)

1.8.1 Visual Paradigm para UML

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Presenta un entorno visual de modelado que logra facilitar el diseño visual de manera dramática. Esta herramienta ayuda a los equipos de desarrollo de software a sobrepasar el proceso de desarrollo de software que incluye modelación-construcción-despliegue, maximizando y acelerando las contribuciones tanto del equipo de desarrollo como las individuales.

Su Generación de Código e Ingeniería Inversa soportan un conjunto de lenguajes como son: Java, C++, CORBA IDL[1], PHP, XML Schema, Ada y Python. En adición a esto, su Generación de Código también soporta otros lenguajes como C#, VB, ODL[2], ActionScript, Delphi, Perl, Objective-C y Ruby. (17)

1.9 Marco de trabajo de desarrollo

El concepto marco de trabajo se emplea en muchos ámbitos del desarrollo de sistemas de software, no solo en el ámbito de aplicaciones web. Se puede encontrar marcos de trabajos para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, entre otros.

En general, con el término marco de trabajo, nos estamos refiriendo a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un marco de trabajo se puede considerar como una

[1] *Interface Definition Language: lenguaje de especificación de interfaces que se utiliza en software de computación distribuida.*

[2] *Object Definition Language: es una extensión de la del IDL, se utiliza para definir las interfaces de los tipos de objetos.*

aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Los objetivos principales que persigue un marco de trabajo son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Un marco de trabajo simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un marco de trabajo proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un marco de trabajo facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

1.9.1 ¿Qué es un marco de trabajo web?

Un marco de trabajo web, podemos definirlo como un conjunto de componentes (por ejemplo clases en Java y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas web. (18).

Facilidades que brinda un marco de trabajo

El programador no necesita plantearse una estructura global de la aplicación, sino que el marco de trabajo le proporciona un esqueleto que hay que “rellenar”.

Facilita la colaboración. Cualquiera que haya tenido que “pelearse” con el código fuente de otro programador (o incluso con el propio, pasado algún tiempo) sabrá lo difícil que es entenderlo y modificarlo; por tanto, todo lo que sea definir y estandarizar va a ahorrar tiempo y trabajo a los desarrollos colaborativos.

Es más fácil encontrar herramientas (utilidades, librerías) adaptadas al marco de trabajo concreto para facilitar el desarrollo.

1.9.2 Symfony

Symfony es un completo marco de trabajo diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web

compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. A continuación se muestran algunas de sus características.

Características de Symfony:

Symfony se diseñó para que se ajustara a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas.
- Independiente del sistema gestor de bases de datos.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de “convenir en vez de configurar”, en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de *phpDocumentor* y que permite un mantenimiento muy sencillo.
- Flexible hasta cualquier límite y extensible mediante un completo mecanismo de *plugins*.
- Publicado bajo licencia MIT de software libre y apoyado por una empresa comprometida con su desarrollo.
- Traducido a más de 40 idiomas y fácilmente traducible a cualquier otro idioma. (19)

1.10 Entorno integrado de desarrollo

Un IDE (acrónimo en inglés de *Integrated Development Environment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. (20)

1.10.1 NetBeans IDE

NetBeans IDE es una aplicación de código abierto ("*open source*") diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología Java.

NetBeans IDE dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis y por si fuera poco sus funcionalidades son ampliables mediante la instalación de paquetes.

NetBeans IDE soporta PHP 5.3 y Symfony, es por esto que lo vamos a utilizar como IDE de desarrollo debido a las facilidades que brinda.

1.11 Lenguaje de programación PHP

Pre-procesador de Hipertexto PHP (del inglés: *Hypertext Preprocessor*) es un lenguaje de programación del lado del servidor que permite crear y ejecutar aplicaciones web dinámicas e interactivas. Es un lenguaje interpretado de propósito general, que está diseñado especialmente para desarrollo web y puede ser introducido dentro de código HTML. Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos sin costo alguno. PHP se encuentra instalado en más de 20 millones de sitios web y en un millón de servidores. Es un lenguaje multiplataforma, tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, posee una amplia documentación en su página oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda, es libre por lo que se presenta como una alternativa de fácil acceso para todos, permite las técnicas de Programación Orientada a Objetos (POO), biblioteca nativa de

funciones sumamente amplia e incluida, no requiere definición de tipos de variables y tiene manejo de excepciones (desde PHP 5). Las principales características para tener en cuenta en un lenguaje script son: velocidad, estabilidad, seguridad y simplicidad. PHP cuenta con cada una de estas características en buena medida:

- Velocidad: no solo la velocidad de ejecución, que es importante, sino también no crear demoras en la máquina. Por esta razón no debe requerir demasiados recursos de sistema. PHP se integra muy bien a otro software, especialmente bajo ambientes Unix y cuando se configura como módulo de Apache.
- Estabilidad: la velocidad no sirve de mucho si el sistema se cae cada cierta cantidad de ejecuciones. Ninguna aplicación es 100% libre de errores, pero teniendo como respaldo una increíble comunidad de programadores y usuarios, es mucho más difícil que los errores sobrevivan. PHP utiliza su propio sistema de administración de recursos y dispone de un sofisticado método de manejo de variables, conformando un sistema robusto y estable.
- Seguridad: el sistema debe poseer protecciones contra ataques. PHP provee diferentes niveles de seguridad que pueden ser configurados desde el archivo (.ini).
- Simplicidad: se les debe permitir a los programadores generar código productivamente en el menor tiempo posible. Usuarios con experiencia en C y C++ podrán utilizar PHP rápidamente. (21)

1.12 JavaScript

JavaScript es el lenguaje *scripting* por excelencia, es decir, es un lenguaje basado en *scripts* (guión o conjunto de instrucciones). Posee una sintaxis similar a la del lenguaje Java y el lenguaje C, aunque no es un lenguaje orientado a objetos propiamente dicho, ya que no dispone de herencia. Está destinado al desarrollo de aplicaciones web como complemento del HTML.

Ventajas:

- No requiere tiempo de compilación.
- Los *scripts* pueden desarrollarse en un período de tiempo relativamente corto.

- Posee características de interfaz, que son gestionados por el navegador y por el código HTML.
- Los programas JavaScript tienden a ser pequeños y compactos, no requieren mucha memoria ni tiempo adicional de transmisión. (22)

1.13 Servidor web

Un servidor web sirve contenido estático a un navegador, carga un archivo y lo sirve a través de la red al navegador de un usuario. Este intercambio es mediado por el navegador y el servidor que hablan el uno con el otro mediante HTTP, es decir, es un software que funciona en un ordenador y maneja la entrega de los componentes de las páginas como respuesta a peticiones de los navegadores de los clientes. Se pueden utilizar varias tecnologías en el servidor para aumentar su potencia más allá de su capacidad de entregar páginas HTML; estas incluyen seguridad SSL entre otras. (23)

1.13.1 Servidor web Apache

Apache es el servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. Apache es una muestra, al igual que el sistema operativo Linux, de que el trabajo voluntario y cooperativo dentro de Internet es capaz de producir aplicaciones de calidad profesional difíciles de igualar. (24)

La licencia Apache es una descendiente de la licencias BSD[3], no es GPL[4]. Esta licencia permite hacer lo que quieras con el código fuente siempre que les reconozcas su trabajo.

Características que hacen popular a este software libre:

- Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Apache es una tecnología gratuita de código fuente abierto. El hecho de ser gratuita es importante pero no tanto como que se trate de código fuente abierto. Esto le da una

[3] *Berkeley Software Distribution*

[4] *GNU General Public License*

transparencia a este software de manera que si se quiere ver qué es lo que se está instalando como servidor, se puede saber sin ningún secreto.

- Apache es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar las capacidades del servidor web Apache. Actualmente existen muchos módulos para Apache que son adaptables a este y están ahí para que se instalen cuando se necesite.
- Otra cosa importante es que cualquiera que posea una experiencia decente en la programación de C o Perl puede escribir un módulo para realizar una función determinada.
- Apache te permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto.
- Tiene una alta configurabilidad en la creación y gestión de *logs*. Apache permite la creación de ficheros de *log* a medida del administrador, de este modo puedes tener un mayor control sobre lo que sucede en tu servidor.

1.14 Sistema gestor de base de datos

PostgreSQL es un gestor de bases de datos objeto-relacional (ORDBMS), es una derivación libre (*OpenSource*) del proyecto POSTGRES, esta versión incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional.

Unas de sus principales características es que soporta distintos tipos de datos, además de los del tipo base. También soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, etc. y permite la creación de tipos propios.

Ventajas:

- Posee una gran escalabilidad. Es capaz de ajustarse al número de Unidades Centrales de Procesamiento (CPUs) y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta.

- Implementa el uso de *rollback*[5], subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz.
- Tiene la capacidad de comprobar la integridad referencial; así como también la de almacenar procedimientos en la propia base de datos.
- El único costo asociado a él, es el de conocerlo pues su código fuente está disponible bajo la más liberal de las licencias del *OpenSource*: la licencia BSD, que permite usarlo, modificarlo y distribuirlo en productos comerciales o no comerciales, sin costo alguno.
- Requiere pocos recursos de hardware y la simplificación del proceso de administración de licencias de software, que no es necesario cuando se usa software libre.
- PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.
- Puede lidiar con gran volumen de datos. (25)

1.15 Conclusiones del capítulo

En este capítulo se realizó un estudio del estado del arte de los sistemas de gestión de activos a nivel internacional y en el país, los cuales solo sirvieron de guía, pues no se adaptan a los requerimientos del sistema que se desea desarrollar. También se analizó la metodología, el lenguaje de modelado, herramientas, lenguaje de programación, el sistema gestor de base de datos y servidor web, exigidos por el cliente a utilizar en el desarrollo de la aplicación, que le dará solución al problema que se planteó en el comienzo de este trabajo, el cual tiene que ver con el desarrollo de un sistema de gestión y control de los activos de Telecomunicaciones de la UCI. Además, se dan explicaciones y justificaciones del por qué de la elección de la metodología FDD como guía para el desarrollo del software; así como PHP 5 como lenguaje de programación, la utilización de Symfony como marco de trabajo para agilizar el desarrollo del sistema, PostgreSQL como sistema gestor de base de datos y Apache como servidor web.

[5] *Rollback*: Operación que devuelve a la base de datos a algún estado previo.

CAPÍTULO II: Características y diseño del sistema

2.1 Introducción

Una vez realizado un análisis sobre la situación actual en la Dirección de Gestión Tecnológica (DGT) de la Universidad de las Ciencias Informáticas, referente a las dificultades que existen en el proceso de manejo y control de los activos de Telecomunicaciones se llega a la conclusión de que es necesario desarrollar un sistema que gestione dichos activos, en el presente capítulo se relacionan un conjunto de funcionalidades que representan las características de la aplicación. Una vez creada la lista de funcionalidades estas se planifican para la posterior fase iterativa de diseño y construcción. Se construyen los diagramas de clases del diseño para cada una de las funcionalidades, se define el estilo arquitectónico y los patrones a utilizar. Se muestra también el modelo de datos propuesto, describiéndose cada una de sus tablas y se representa el modelo de despliegue.

2.2 Descripción de la aplicación

Para la gestión y control de los activos de Telecomunicaciones se necesita un sistema que informatice las actividades y procesos que se llevan a cabo en la Dirección de Gestión Tecnológica, con el propósito de gestionar y controlar los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas. Para solucionar toda la problemática que se plantea en la investigación, la aplicación organiza los procesos que se llevan a cabo en la DGT en cuatro módulos. Además, gestiona la administración del sistema con un nuevo módulo, garantizando su seguridad; así como un conjunto de elementos que le brindan al sistema mayores funcionalidades.

2.3 Descripción de procesos

Los procesos de negocio son un conjunto estructurado de actividades, diseñado para producir una salida determinada o lograr un objetivo. Estos describen cómo es realizado el trabajo y se caracterizan por ser observables, medibles, mejorables y repetitivos. (26)

Luego de realizar un estudio y análisis mediante entrevistas sobre los procesos que se llevan a cabo en la Dirección de Gestión Tecnológica sobre la gestión y control de los activos de Telecomunicaciones en la UCI se identificaron un conjunto de procesos que se desean automatizar, estos se describen a continuación.

Registrar conmutador:

Los conmutadores se recogen en el local de tránsito de la Dirección de Gestión Tecnológica, posteriormente son registrados en una planilla **Actualización de Conmutador**, en dicha planilla se recogen todos los datos del conmutador; los datos que se registran en dicho proceso son: número de serie, número de inventario, ubicación, el estado en que se encuentra el conmutador, tipo de conmutador, fabricante, modelo, dirección ip, cantidad de puertos par trenzado no apantallado o *Unshielded Twisted Pair* (UTP), velocidad de los puertos UTP, cantidad de puertos de fibra óptica (FO), velocidad de los puertos FO, fecha de compra, fecha de vencimiento, fecha de salida y observaciones, la planilla se almacena en uno de los archivos de la DGT. Este proceso lo lleva a cabo el Técnico General del grupo de conectividad.

Modificar conmutador:

Estos conmutadores se pueden mover de lugar ya sea por rotura o por la necesidad de hacer algún tipo de mantenimiento o préstamo, lo que trae consigo una modificación en los datos iniciales, todos estos cambios son actualizados en la planilla **Actualización de Conmutador,** la planilla se almacena en uno de los archivos de la DGT. Este proceso lo lleva a cabo el Técnico General del grupo de conectividad.

Eliminar conmutador:

Cuando un conmutador deja de funcionar el Técnico General del grupo de conectividad notifica a Copextel de la rotura, si este valora que su ruptura es irreparable seguidamente se elimina de la planilla **Actualización de Conmutador.**

Registrar Beeper:

Los beeper son comprados por el Grupo Económico de la Dirección de Gestión Tecnológica y otorgados al personal por dicha dirección, esto se registra en una planilla llamada **Beepers UCI MOVITEL**, donde se recogen los siguientes datos: número de beeper, responsable (nombre y cargo), estado del beeper, área a la que pertenece, tipo de servicio y

observaciones. La planilla se guarda en un archivo de la DGT. Este proceso lo lleva a cabo el Especialista General del grupo de comunicaciones.

Modificar Beeper:

Los beeper pueden sufrir modificaciones como: cambio de responsable, servicio (activado o cancelado), tipo de servicio (interno o nacional) y estado (activo, recogido, roto, MOVITEL). Al producirse alguna alteración en los datos de este, se actualizan los cambios en la planilla **Beepers UCI MOVITEL**. . La planilla se guarda en un archivo de la DGT. Este proceso lo lleva a cabo el Especialista General del grupo de comunicaciones.

Eliminar Beeper:

En caso de rotura de un beeper, estos son llevados a MOVITEL para su reparación, si MOVITEL estima que no tiene solución, dicho beeper es eliminado de la planilla **Beepers UCI MOVITEL**. . Este proceso lo lleva a cabo el Especialista General del grupo de comunicaciones.

Registrar Planta:

Las plantas son compradas a MOVITEL por el Grupo Económico de la Dirección de Gestión Tecnológica al igual que los beeper, una vez asignadas estas plantas, se hace un acta de entrega y se registran todos los datos en una planilla llamada **Plantas,** en la cual se almacenan todos los datos de esta como son: estado de la planta, tipo de planta, localización, modelo, área, responsable, flota, número de la planta, grupo, si tiene servicio de teléfono o no, en caso de tener, almacenar el número de teléfono, servicio y observaciones. . La planilla se guarda en un archivo de la DGT. Este proceso lo lleva a cabo el Especialista General del grupo de comunicaciones.

Modificar Planta:

Las plantas pueden sufrir modificaciones como: cambio de estado, localización, área, responsable, servicio (activado o cancelado), servicio de teléfono, número de planta, grupo, además están expensas a roturas. Cualquier tipo de cambio que ocurra con estas plantas será registrado en la planilla **Plantas**. . La planilla se guarda en un archivo de la DGT. Este proceso lo lleva a cabo el Especialista General del grupo de comunicaciones.

Eliminar Planta:

En caso de rotura de una planta, si MOVITEL no tiene solución para esta, entonces se elimina de la planilla **Plantas**. Este proceso lo lleva a cabo el Especialista General del grupo de comunicaciones.

Registrar Pin:

Los pines son generados en la pizarra de telefonía de la Dirección de Gestión Tecnológica y son asignados a un listado de cargos de dirección, todos estos datos son registrados en la planilla llamada **PIN_UCI**, los datos que se registran son: pin, responsable del pin, área a la que pertenece el pin, cuota, correo del responsable, teléfono del responsable y observaciones. La planilla se guarda en un archivo de la DGT. Este proceso lo lleva a cabo el Especialista General del grupo de conectividad.

Modificar Pin:

Los pines sufren modificaciones solamente cuando existe un cambio de responsable, al ocurrir un cambio de responsable el pin cambia, también existe la posibilidad de que un responsable de un pin no se aprenda el número y se vea en la necesidad de solicitar a la Dirección de Gestión Tecnológica un cambio en el número de su pin, todos estos datos son actualizados en la planilla **PIN_UCI.** La planilla se guarda en un archivo de la DGT. Este proceso lo lleva a cabo el Especialista General del grupo de conectividad.

Eliminar Pin:

Un pin es eliminado cuando este es usado indebidamente, cuando el pin tiene sobreuso (gasto de la cuota del pin en los primeros 5 días del mes) todos los meses y cuando el número del pin pasa a ser de conocimiento para otras personas ajenas al cargo, en caso de ocurrir esto el pin es eliminado de la planilla **PIN_UCI**. Este proceso lo lleva a cabo el Especialista General del grupo de conectividad.

2.4 Modelo global del sistema

El Modelo Global del Sistema consiste en la construcción de un diagrama de clases que representa los tipos de objetos más importantes dentro del dominio del problema y las relaciones entre ellos. Este diagrama de clases es de carácter estructural y luce como el tradicional diagrama entidad-relación de las bases de datos relacionales, pero presenta dos

grandes diferencias ya que puede incluir relaciones de herencia, generalización y especialización, y las operaciones no reflejan conveniencias de programación sino que se describen en formas de funcionalidades especificando cómo debe comportarse el objeto. (27)

A continuación se muestra en la figura No. 2 el Modelo Global del Sistema desarrollado.

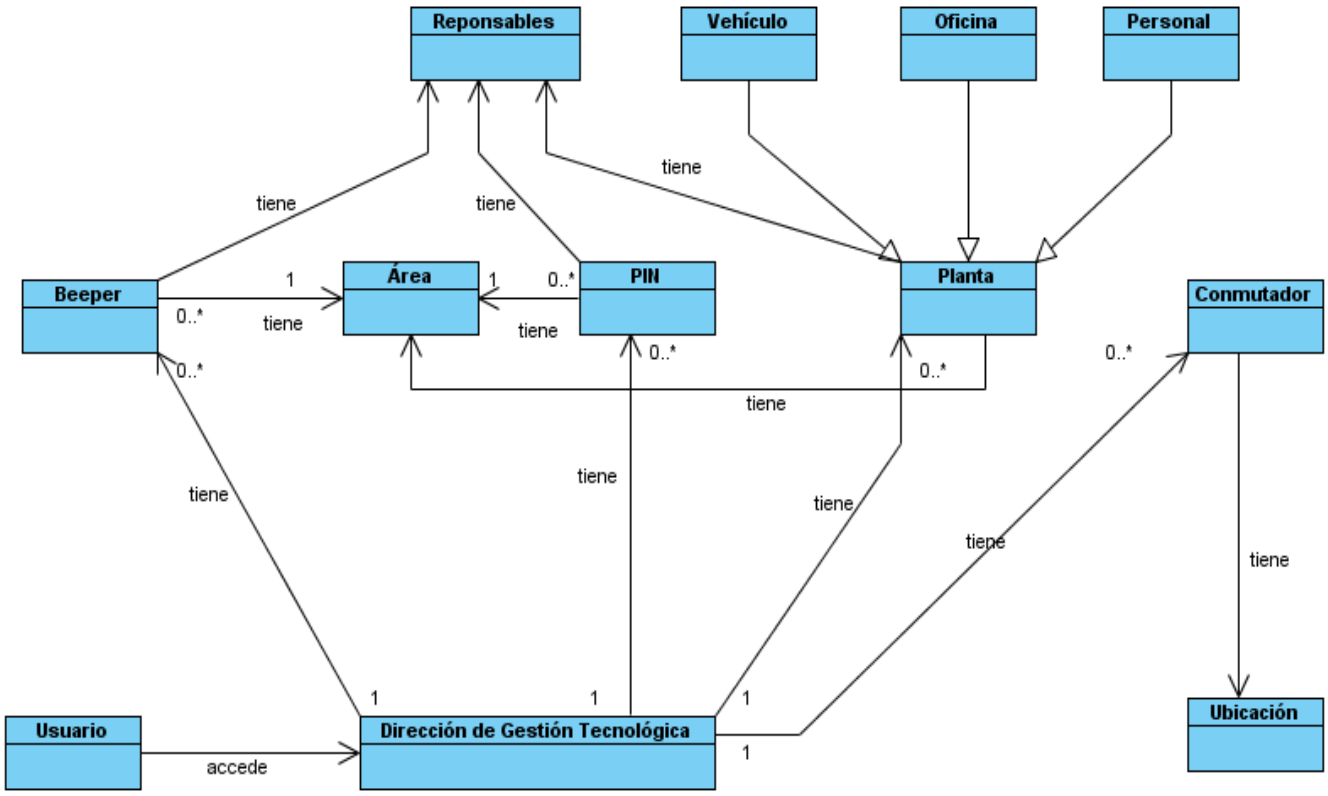


Figura 2. Modelo Global del Sistema

2.5 Construcción de la lista de funcionalidades

El modelo global y la documentación de requerimientos proporcionan la base para construir una amplia lista de funcionalidades. Estas funcionalidades son pequeños detalles útiles a los ojos del cliente. La lista de funcionalidades es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad.

A continuación se muestra en la tabla No. 1, la lista de funcionalidades que tendrá el sistema a desarrollar:

Lista de funcionalidades		
Área Temática	**Actividades del Negocio**	**Funcionalidades**
Administración	Gestionar usuario.	Permitir autenticar usuario. Permitir adicionar un nuevo usuario. Permitir la asignación de roles. Permitir la asignación de permisos. Buscar usuarios que coincidan con criterios de búsqueda y mostrar un listado de ellos**.** Modificar cuenta de usuario mostrando los datos del usuario y mostrar un mensaje confirmando la modificación. Eliminar cuenta de usuario mostrando un mensaje confirmando la eliminación.
Gestión de conmutador	Gestionar conmutador.	Registrar un conmutador y mostrar un mensaje confirmando la inserción. Buscar conmutadores dados criterios de búsqueda y mostrar un listado de ellos. Modificar conmutador mostrando sus datos y enviar un mensaje confirmando la modificación. Mostrar un listado de los conmutadores existentes. Eliminar conmutador y mostrar mensaje confirmando la eliminación.
Gestión de beeper	Gestionar beeper	Registrar un beeper y mostrar un mensaje confirmando la inserción. Buscar beeper dados criterios de búsqueda y mostrar un listado de ellos. Modificar beeper mostrando sus datos y enviar mensaje confirmando la modificación. Mostrar un listado de los beeper existentes. Eliminar beeper y mostrar un mensaje confirmando la eliminación.

Gestión de planta	Gestionar planta	Registrar una planta y mostrar un mensaje confirmando la inserción. Buscar planta dados criterios de búsqueda y mostrar un listado de ellos. Modificar planta mostrando sus datos y enviar un mensaje confirmando la modificación. Mostrar un listado de las plantas existentes. Eliminar planta y mostrar un mensaje confirmando la eliminación.
Gestión de pin	Gestionar pin	Registrar un pin y mostrar un mensaje confirmando la inserción. Buscar pin dados criterios de búsqueda y mostrar un listado de ellos. Modificar pin mostrando sus datos y enviar un mensaje confirmando la modificación. Mostrar un listado de los pines existentes. Eliminar pin y mostrar mensaje confirmando la eliminación.
Gestión de reportes	Generar reportes	Permitir al usuario generar reportes de conmutadores según el criterio deseado brindando la posibilidad exportarlo como PDF. Permitir al usuario generar reportes de beepers según el criterio deseado brindando la posibilidad de exportarlo como PDF. Permitir al usuario generar reportes de plantas según el criterio deseado brindando la posibilidad de exportarlo como PDF. Permitir al usuario generar reportes de pines según el criterio deseado brindando la posibilidad de exportarlo como PDF.

Tabla 1. Lista de funcionalidades

2.6 Planeación por funcionalidades

Antes de empezar a diseñar y construir la aplicación se realizó una planeación de la lista de funcionalidades identificada. Esto se planificó teniendo en cuenta todas las funcionalidades identificadas en el proceso anterior. A la hora de planear cada funcionalidad se tuvo en cuenta que dentro de una iteración no pueden existir funcionalidades que dependan de otras funcionalidades que no hayan sido implementadas, con la excepción de que puede depender de funcionalidades no implementadas, pero que todas estas se encuentren en la misma iteración. A continuación se muestran en forma de tablas la planeación de cada funcionalidad.

Gestionar conmutador				**Funcionalidades: 5**	
Descripción	**Modelo Global**	**Diseño**	**Construcción**	**Prueba**	**Despliegue**
	Plan	**Plan**	**Plan**	**Plan**	**Plan**
Registrar un conmutador y mostrar un mensaje confirmando la inserción.	**25/01/10**	**01/02/10**	**04/02/10**	**11/02/10**	**10/05/10**
Buscar conmutador dados criterios de búsqueda y mostrar un listado de ellos.	**25/01/10**	**01/02/10**	**04/02/10**	**11/02/10**	**10/05/10**
Modificar conmutador mostrando sus datos y enviar un mensaje confirmando la modificación.	**25/01/10**	**01/02/10**	**04/02/10**	**11/02/10**	**10/05/10**
Mostrar un listado de los conmutadores existentes.	**25/01/10**	**01/02/10**	**04/02/10**	**11/02/10**	**10/05/10**
Eliminar conmutador y mostrar un mensaje confirmando la eliminación.	**25/01/10**	**01/02/10**	**04/02/10**	**11/02/10**	**10/05/10**

Tabla 2. Planificación de Gestionar conmutador

Gestionar pin				Funcionalidades: 5	
Descripción	**Modelo Global**	**Diseño**	**Construcción**	**Prueba**	**Despliegue**
	Plan	**Plan**	**Plan**	**Plan**	**Plan**
Registrar un pin y mostrar un mensaje confirmando la inserción.	**25/01/10**	**15/02/10**	**18/02/10**	**25/02/10**	**10/05/10**
Buscar pin dados criterios de búsqueda y mostrar un listado de ellos.	**25/01/10**	**15/02/10**	**18/02/10**	**25/02/10**	**10/05/10**
Modificar pin mostrando sus datos y enviar un mensaje confirmando la modificación.	**25/01/10**	**15/02/10**	**18/02/10**	**25/02/10**	**10/05/10**
Mostrar un listado de los pines existentes.	**25/01/10**	**15/02/10**	**18/02/10**	**25/02/10**	**10/05/10**
Eliminar pin y mostrar un mensaje confirmando la eliminación.	**25/01/10**	**15/02/10**	**18/02/10**	**25/02/10**	**10/05/10**

Tabla 3. Planificación de Gestionar pin

Gestionar beeper				Funcionalidades: 5	
Descripción	**Modelo Global**	**Diseño**	**Construcción**	**Prueba**	**Despliegue**
	Plan	**Plan**	**Plan**	**Plan**	**Plan**
Registrar un beeper y mostrar un mensaje confirmando la inserción.	**25/01/10**	**01/03/10**	**04/03/10**	**10/03/10**	**10/05/10**
Buscar beeper dados criterios de búsqueda y mostrar un listado de ellos.	**25/01/10**	**01/03/10**	**04/03/10**	**10/03/10**	**10/05/10**
Modificar beeper mostrando sus datos y enviar un	**25/01/10**	**01/03/10**	**04/03/10**	**10/03/10**	**10/05/10**

mensaje confirmando la modificación.					
Mostrar un listado de los beeper existentes.	**25/01/10**	**01/03/10**	**04/03/10**	**10/03/10**	**10/05/10**
Eliminar beeper y mostrar un mensaje confirmando la eliminación.	**25/01/10**	**01/03/10**	**04/03/10**	**10/03/10**	**10/05/10**

Tabla 4. Planificación de Gestionar beeper

Gestionar planta.				**Funcionalidades: 5**	
Descripción	**Modelo Global**	**Diseño**	**Construcción**	**Prueba**	**Despliegue**
	Plan	**Plan**	**Plan**	**Plan**	**Plan**
Registrar una planta y mostrar un mensaje confirmando la inserción.	**27/01/10**	**15/03/10**	**18/03/10**	**25/03/10**	**10/05/10**
Buscar planta dados criterios de búsqueda y mostrar un listado de ellos.	**27/01/10**	**15/03/10**	**18/03/10**	**25/03/10**	**10/05/10**
Modificar planta mostrando sus datos y enviar un mensaje confirmando la modificación.	**27/01/10**	**15/03/10**	**18/03/10**	**25/03/10**	**10/05/10**
Mostrar un listado de las plantas existentes.	**27/01/10**	**15/03/10**	**18/03/10**	**25/03/10**	**10/05/10**
Eliminar planta y mostrar un mensaje confirmando la eliminación.	**27/01/10**	**15/03/10**	**18/03/10**	**25/03/10**	**10/05/10**

Tabla 5. Planificación de Gestionar Planta

Generar reportes				Funcionalidades: 4	
Descripción	**Modelo Global**	**Diseño**	**Construcción**	**Prueba**	**Despliegue**
	Plan	**Plan**	**Plan**	**Plan**	**Plan**
Permitir al usuario generar reportes de conmutadores según el criterio deseado brindando la posibilidad de exportarlo como PDF.	**27/01/10**	**29/03/10**	**01/04/10**	**08/04/10**	**10/05/10**
Permitir al usuario generar reportes de beepers según el criterio deseado brindando la posibilidad de exportarlo como PDF.	**27/01/10**	**29/03/10**	**01/04/10**	**08/04/10**	**10/05/10**
Permitir al usuario generar reportes de plantas según el criterio deseado brindando la posibilidad de exportarlo como PDF.	**27/01/10**	**29/03/10**	**01/04/10**	**08/04/10**	**10/05/10**
Permitir al usuario generar reportes de pines según el criterio deseado brindando la posibilidad de exportarlo como PDF.	**27/01/10**	**29/03/10**	**01/04/10**	**08/04/10**	**10/05/10**

Tabla 6. Planificación de Generar reportes

Gestionar usuario				Funcionalidades: 5	
Descripción	**Modelo Global**	**Diseño**	**Construcción**	**Prueba**	**Despliegue**
	Plan	**Plan**	**Plan**	**Plan**	**Plan**
Permitir autenticar usuario.	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**
Permitir adicionar un nuevo	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**

usuario.					
Permitir la asignación de roles.	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**
Permitir la asignación de permisos.	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**
Buscar usuarios que coincidan con criterios de búsqueda y mostrar un listado de los ellos**.**	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**
Modificar cuenta de usuario mostrando los datos del usuario y mostrar un mensaje confirmando la modificación.	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**
Eliminar cuenta de usuario mostrando un mensaje confirmando la eliminación.	**27/01/10**	**12/04/10**	**15/04/10**	**22/04/10**	**10/05/10**

Tabla 7. Planificación de Gestionar usuario

2.7 Requisitos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener, y son los que harán de la aplicación un producto atractivo, usable, rápido y confiable.

RNF 1. **Apariencia o interfaz externa:**

- La aplicación contará con una interfaz sencilla y con colores agradables a la vista como los distintos tonos de azules, blancos y algunas tonalidades de gris. Hará uso de banners discretos y un mapa de navegación cómodo para el usuario.

RNF 2. **Usabilidad:**

- El sistema podrá ser usado de forma fácil por cualquier persona, aunque será utilizado en su mayoría por el personal de la Dirección de Gestión Tecnológica, pues tendrá una interfaz de manejo cómodo que posibilite a los usuarios sin experiencia una rápida adaptación.

RNF 3. **Especificación de la terminología utilizada**:

- El sistema debe adaptarse al lenguaje y términos utilizados por los clientes en la rama abordada con vista a una mayor comprensión por parte del cliente de la herramienta de trabajo.

RNF 4. Rendimiento:

- El sistema será implementado bajo el paradigma de la web 2.0 que establece como característica fundamental la interactividad entre la aplicación y el usuario, de este modo, el tiempo de respuesta debe ser el menor posible, haciendo uso a la vez de páginas dinámicas que permitan un rápido acceso a la información y de la manera más fácil posible.
- El sistema contará con una base de datos en 3era forma normal (3FN) que garantiza el rendimiento buen rendimiento de la misma.
- La aplicación permitirá que múltiples usuarios estén conectados a la vez.

RNF 5. Soporte:

- El sistema será probado, instalado y configurado por los miembros de la Dirección de Gestión Tecnológica y un Administrador, el cual se ocupará también de su mantenimiento.

RNF 6. Portabilidad:

- Este producto podrá ser utilizado tanto en Windows como en Linux, ya que en su implementación se utilizarán herramientas multiplataforma.

RNF 7. Seguridad:

- Autenticación basada en dos niveles: el sistema debe tener la posibilidad de autenticarse en un Directorio Activo y en caso que este no exista, se podrá hacer directamente en la base de datos local.
- Dado que el sistema se pondrá en práctica dentro de la Universidad de las Ciencias Informáticas, sus mecanismos de seguridad estarán en perfecta consonancia con las políticas de seguridad establecidas en la universidad para garantizar la confidencialidad, integridad y disponibilidad de los datos que se manejan.

- Confidencialidad: la información que se maneja estará protegida de acceso no autorizado, ya que será requerida la autenticación de los usuarios para garantizar que solo las personas autorizadas puedan acceder a estos datos.
- Integridad: la información privada del sistema será objeto de cuidadosa protección contra la corrupción y estados de inconsistencia, ya que solo el personal autorizado será el único que tendrá acceso a realizar algún tipo de cambio.
- Disponibilidad: la aplicación estará disponible en todo momento para aquellas personas con acceso a la información, y los mecanismos utilizados para lograr la seguridad no serán un obstáculo a los usuarios para obtener los datos deseados en el momento que lo requieran.

RNF 8. Confiabilidad:

- La información almacenada, procesada y generada por el sistema será confiable, ya que a las carpetas donde se almacenan los documentos; así como a la base de datos, se le harán copias de seguridad diarias como medida preventiva ante los fallos que pudieran ocurrir.

RNF 9. Software:

- Para el cliente: sistema operativo con interfaz gráfica y conexión a red.
- Navegador Web: Mozilla Firefox (Recomendado).
- Para el servidor: Sistema Operativo Linux Ubuntu 7.10, Windows XP o superior.
- Servidor Web: Apache 2.2.9.
- Gestor de Base de Datos: PostgreSQL 8.3.5.
- Software controlador de versiones: *Subversion* v_1.4.4 (r25188).

RNF 10. Restricciones de diseño

- Lenguaje de programación: PHP 5.3.
- El marco de trabajo de desarrollo que se utilizará es: Symfony 1.4.
- Como IDE se empleará NetBeans 6.8
- El modelado UML se hará con *Visual Paradigm* 3.4.
- El sistema operativo a utilizar en el entorno de desarrollo deberá ser: Windows XP SP 2 ó Ubuntu 7.10 (o superior).

RNF 11. Hardware:

- Para la PC cliente se requiere una máquina con 128 MB de RAM como mínimo, el servidor web, al igual que el servidor de base de datos, debe tener 512 MB de RAM y 60 GB de disco duro mínimo, todas las máquinas implicadas en la funcionalidad de la aplicación deben estar conectadas a la red de al menos 100 Mbps de velocidad.

RNF 12. Ayuda y documentación en línea:

- El sistema contará con mensajes de ayuda que indiquen qué operación realiza cada componente.

2.8 Especificaciones de la arquitectura

La arquitectura de software es una vista estructural de alto nivel, ocurre muy tempranamente en el ciclo de vida y define los estilos o grupos de estilos adecuados para cumplir con los requerimientos no funcionales. (28).

2.8.1 Arquitectura Cliente/Servidor

La arquitectura Cliente/Servidor es una tendencia en el desarrollo de redes, que tiene como objetivo optimizar el uso tanto del hardware como del software, a través de la separación de funciones: el cliente, quien inicia una determinada petición y el servidor, dedicado a responder dichas peticiones, como se muestra en la figura No. 3.

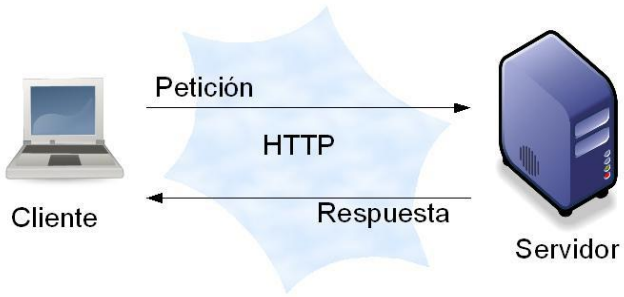


Figura 3. Arquitectura Cliente/Servidor

Puede presentarse como uno o varios clientes y servidores, junto con un sistema operativo y una plataforma de comunicación para formar un sistema cooperativo que permita la computación distribuida, el análisis y la presentación de datos. Un único servidor típicamente

sirve a una multitud de clientes, ahorrando a cada uno de ellos el problema de tener la información almacenada localmente. (29).

Características de la arquitectura Cliente/Servidor:

- El servidor presenta una interfaz única y bien definida a todos sus clientes.
- El cliente no necesita conocer la lógica del servidor, solo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor no afectan al cliente.

2.8.2 El patrón Modelo Vista Controlador (MVC)

Symfony está basado en un patrón clásico del diseño web conocido como arquitectura MVC, que está formado por tres niveles:

- El Modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.
- La Vista transforma el modelo en una página web que permite al usuario interactuar con ella.
- El Controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. (30)

El principio más importante de la arquitectura MVC es la separación del código de la aplicación en tres capas, dependiendo de su naturaleza. La lógica relacionada con los datos se incluye en el modelo, el código de la presentación en la vista y la lógica de la aplicación en el controlador.

Modelo: es la representación de la información que maneja la aplicación. El modelo en sí son los datos puros que puestos en contexto del sistema proveen de información al usuario y a la aplicación misma.

Vista: es la representación del modelo en forma gráfica, disponible para la interacción con el usuario. En el caso de una aplicación web, la “Vista” es una página HTML con contenido dinámico sobre el cual el usuario puede realizar operaciones.

Controlador: es la capa encargada de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el modelo en caso de ser necesario.

Contenido de cada capa en Symfony:

La capa del Modelo

- Abstracción de la base de datos
- Acceso a los datos

La capa de la Vista

- Vista
- Plantilla
- Layout

La capa del Controlador

- Controlador frontal
- Acción

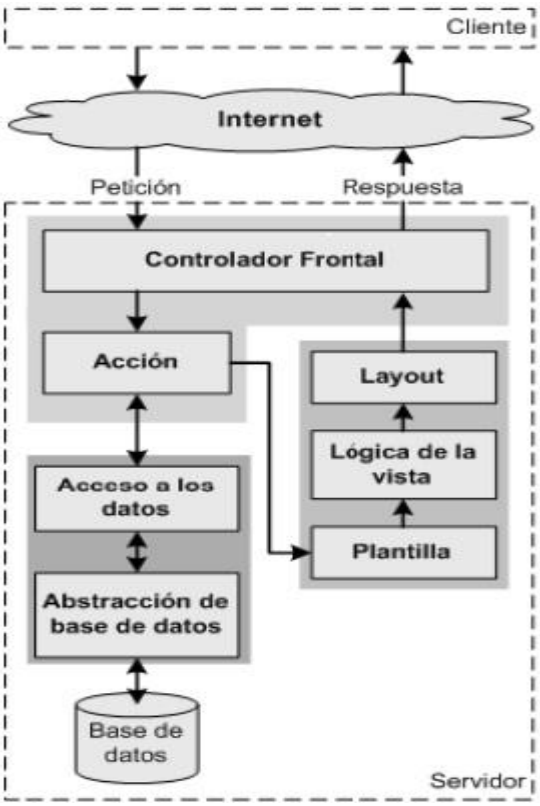


Figura 4. Flujo de trabajo de Symfony

Ventajas y desventajas de MVC.

Las principales ventajas del MVC son:

- La separación del Modelo de la Vista, es decir, separar los datos de la representación visual de los mismos.
- Es mucho más sencillo agregar múltiples representaciones de los mismos datos.
- Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento.
- Facilita el mantenimiento en caso de errores.
- Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema.

Las desventajas:

- La separación de conceptos en capas agrega complejidad al sistema.
- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.
- El aprendizaje del patrón es más lento que otros modelos más sencillos.

2.9 Patrones de diseño utilizados

El desarrollo del sistema utilizando el marco de trabajo Symfony proporciona ventajas significativas para los desarrolladores de software. El marco de trabajo mencionado es capaz de fusionar buenas prácticas de trabajo por sí mismo, de forma que los desarrolladores no tengan que preocuparse por implementar varios de los patrones arquitectónicos y de diseño más utilizados en la actualidad, ya que el mismo marco de trabajo los implementa.

2.9.1 Patrones GRASP implementados

Creador: todos los módulos del sistema tienen una clase *actions.class.php* que contiene las acciones definidas para dichos módulos y es en ella misma donde se ejecutan las funciones que hacen al sistema funcional. En esta clase las acciones se encargan de crear los objetos de las clases que representan las entidades, evidenciando de este modo que la clase *actions.class.php* es el "*creador*" de las entidades.

Controlador: todas las peticiones web son manejadas por un solo controlador frontal (*frontend_dev.php*), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL entrada por el usuario.

Experto: se evidencia este patrón puesto que Doctrine es la librería externa que utiliza Symfony para realizar su capa de abstracción al modelo de datos, encapsulando toda la lógica de los datos y generando las clases con funcionalidades comunes de las entidades. Por tanto cada clase creada por Doctrine a partir de una entidad es experta en manejar su información.

Alta Cohesión: Symfony permite la asignación de responsabilidades con alta cohesión, por ejemplo la clase *actions.class.php* tiene la responsabilidad para definir las acciones sobre las plantillas y colabora con otras para realizar diferentes operaciones y crear objetos, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas, proporcionando que el software sea flexible frente a grandes cambios.

2.9.2 Patrones GOF utilizados

***Singlenton* (Instancia Única):** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Es el caso del controlador frontal, donde hay una llamada a la función sfContext::getInstance() que garantiza que siempre se acceda a la misma instancia.

***Decorator* (Decorador):** añade funcionalidad a una clase, dinámicamente. El archivo *layout.php*, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el *layout* decorando la misma.

***Abstrac Factory* (Fábrica Abstracta):** se utiliza este patrón al trabajar con objetos de distintas familias de manera que no se mezclen entre sí, haciendo transparente el tipo de familia concreta que se esté usando. Cuando el marco de trabajo necesita, por ejemplo, crear un nuevo objeto, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.

2.10 Clases del diseño

"Una clase del diseño es una abstracción sin costuras de una clase o construcción similar en la implementación del sistema". (31)

2.10.1 Extensiones para el diseño web

Las extensiones UML para el diseño web, exponen la solución para el modelamiento de diagramas de clases del diseño sobre tecnologías web. De forma que quedan presentados los siguientes estereotipos.

Estereotipos	Imagen	Descripción
Server Page		Representa una página web dinámica que contiene el código ensamblado por el servidor cada vez que se solicita. Típicamente, una página servidora contiene *scripts* que se ejecutan en el servidor y que actúan recíprocamente con los recursos del lado del servidor estos son: las bases de datos, los

		componentes de la lógica del negocio y sistemas externos.
Client Page		Representa una instancia de una página cliente. Es una página web con formato HTML. Las páginas cliente son interpretadas y mostradas por los navegadores del cliente y además pueden contener *scripts* que se interpretan en el navegador.
Form		Simboliza un formulario, es el elemento encargado de realizar envíos a las páginas servidoras.

Tabla 8. Estereotipos web

Relaciones que se establecen entre las clases que conforman la extensión UML para web.

Hasta-Desde	***Server Page***	***Client Page***	***Form***
Server Page	*<<Redirect>>*	*<<Build>>*, *<<Redirect>>*	--
Client Page	*<<Link>>*, *<<Redirect>>*	*<<Link>>*, *<<Redirect>>*	Contiene
Form	*<<Submit>>*	Agregado por.	--

Tabla 9. Relaciones entre clases

El autor Jim Conallen en su libro "*Building Web Applications with UML Second Edition*", presenta las siguientes descripciones para los estereotipos utilizados en las relaciones entre las clases.

Estereotipo	**Descripción**
<<Link>>	Representa una relación entre una página del cliente y un recurso del lado del servidor, o una página web.
<<Build>>	Representa una relación direccional entre una página servidora y cliente. Esta relación identifica la salida HTML de la ejecución de una página servidora.
<<Submit>>	Relación directa entre un formulario <<HTML *form*>> y una página servidora. Similar a la relación <<Link>>, pero solo referencia a recursos del lado servidor. Sin embargo, cuando los recursos son pedidos desde el servidor, todos los campos del formulario son enviados al servidor junto con la petición, donde son procesados.
<<Redirect>>	Relación direccional entre páginas clientes, páginas servidoras y unas con otras.

	Esta asociación indica un comando a la página cliente para realizar petición de otro recurso.
<<*Include*>>	Asociación direccional desde una <<*server page*>> a otra <<*server page*>> o <<*client page*>>. Esta asociación indica que las páginas incluidas son procesadas mientras que la página se ensambla.

Tabla 10. Descripción de las relaciones

2.10.2 Clases del diseño con estereotipos web

Una clase del diseño "completa", es aquella suficientemente detallada que sirve como base para generar código fuente. Una clase del diseño es una clase cuya especificación es completa hasta un nivel que se pueda implementar. (32)

A continuación se muestra en la figura No. 5 el diagrama de clases del diseño del módulo "Beeper", unos del los principales del sistema, los demás están anexados en el documento.

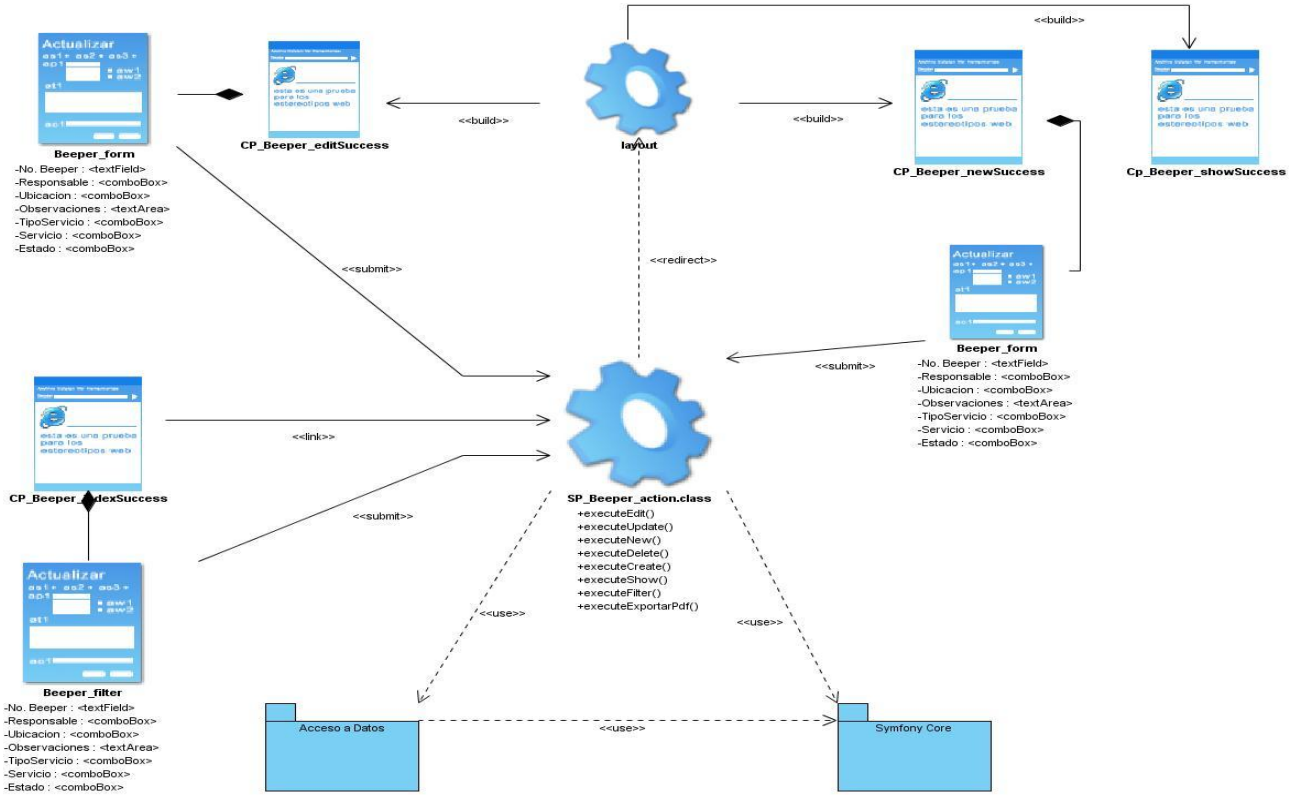


Figura 5. Diagrama de clase del diseño del módulo Beeper

2.10.3 Descripción de las clases del diseño

En esta sección se describen todas las clases del diseño presentadas anteriormente en el diagrama de clase del diseño, para cada una de ellas se muestra su nombre y tipo de clase y el nombre y tipo de cada uno de sus atributos. Se expone además efímeramente el objetivo y forma de funcionamiento de todas las operaciones contenidas por la clase, exceptuando aquellas comúnmente conocidas como "*get*" y "*set*". A continuación se muestra la descripción de una de las principales clases del diseño **"Clases del diseño del módulo Beeper",** el resto puede ser consultado en los anexos del presente trabajo.

Nombre: Beeper_form	
Tipo de clase: Interface	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	use_stylesheets_for_form(()
Descripción:	Carga la hoja de estilo del formulario beeper.
Nombre:	use_javascripts_for_form(()
Descripción:	Carga el script del formulario beeper.
Nombre:	renderHiddenFields()
Descripción:	Muestra los campos ocultos.
Nombre:	renderLabel()
Descripción:	Muestra el campo *label*.
Nombre:	renderGlobalErrors()
Descripción:	Muestra errores globales.
Nombre:	renderError()
Descripción:	Muestra el campo mensajes de error en caso de haber.
Nombre:	image_tag()
Descripción:	Función que permite imprimir una imagen.
Nombre:	link_to()
Descripción:	Función que permite hacer un hipervínculo.
Nombre:	isNew()

Descripción:	Función que verifica si un elemento existe.

Tabla 11. Descripción de la Cl Beeper_form

Nombre: Beeper_editSuccess	
Tipo de clase: Interface	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	include_partial()
Descripción:	Función que incluye un elemento parcial.

Tabla 12. Descripción de la Cl Beeper_editSuccess

Nombre: Beeper_indexSuccess	
Tipo de clase: Interface	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	url_for()
Descripción:	Transforma una url interna a una externa.
Nombre:	count()
Descripción:	Función que permite contar los elementos de un arreglo.
Nombre:	haveToPaginate()
Descripción:	Función que verifica si existe paginado.
Nombre:	image_tag()
Descripción:	Función que permite imprimir una imagen.
Nombre:	renderHiddenFields()
Descripción	Muestra los campos ocultos.
Nombre:	link_to()
Descripción	Función que permite hacer un hipervínculo.

Tabla 13. Descripción de la Cl Beeper_indexSuccess

Nombre: Beeper_newSuccess	
Tipo de clase: Interface	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	include_partial()
Descripción:	Función que incluye un elemento parcial.

Tabla 14. Descripción de la CI Beeper_newSuccess

Nombre: Beeper_showSuccess	
Tipo de clase: Interface	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	url_for()
Descripción:	Transforma una url interna a una externa.
Nombre:	link_to
Descripción:	Función que permite hacer un hipervínculo
Nombre:	image_tag()
Descripción:	Función que permite imprimir una imagen.

Tabla 15. Descripción de la CI Beeper_showSuccess

Nombre: Beeper_actions.class	
Tipo de clase: Controladora	
Atributo	Tipo
Para cada responsabilidad:	
Nombre:	executeIndex()
Descripción:	Muestra la lista de los elementos existentes.
Nombre:	executeFilter()
Descripción:	Función que permite filtrar la lista según un criterio de búsqueda.

Nombre:	executeShow()
Descripción:	Muestra los detalles de los elementos seleccionados
Nombre:	executeNew()
Descripción:	Función que permite crear un nuevo elemento(conjunto a executeCreate())
Nombre:	executeCreate()
Descripción:	Función que permite crear un nuevo elemento(conjunto a executeNew())
Nombre:	executeEdit()
Descripción:	Función que permite editar un elemento(conjunto al executeUpdate())
Nombre:	executeUpdate()
Descripción:	Función que permite editar un elemento(conjunto al executeEdit())
Nombre:	executeDelete()
Descripción:	Función que permite borrar un elemento.
Nombre:	processForm()
Descripción:	Función que utilizan los métodos *create*() y *update*() para procesar el formulario (validación, volver a mostrar los datos del formulario y guardado).
Nombre:	executeExportarPDF()
Descripción:	Función que permite exportar como PDF una lista.
Nombre:	buildQuery()
Descripción:	Función que permite construir una consulta.

Tabla 16. Descripción de la CC Beeper_actions.class

2.11 Modelo de la Base de Datos

Una base de datos es un conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una base de datos puede considerarse una colección de datos variables en el tiempo. (33)

El diagrama de clases persistentes es obtenido a partir de los diagramas de clases del diseño. La persistencia es la propiedad de los objetos de trascender su estado en el tiempo y el espacio. Este diagrama y la descripción de las tablas del mismo se encuentran anexados en el documento. A continuación se muestra en la figura No. 6 el diagrama entidad-relación utilizado en la propuesta de solución.

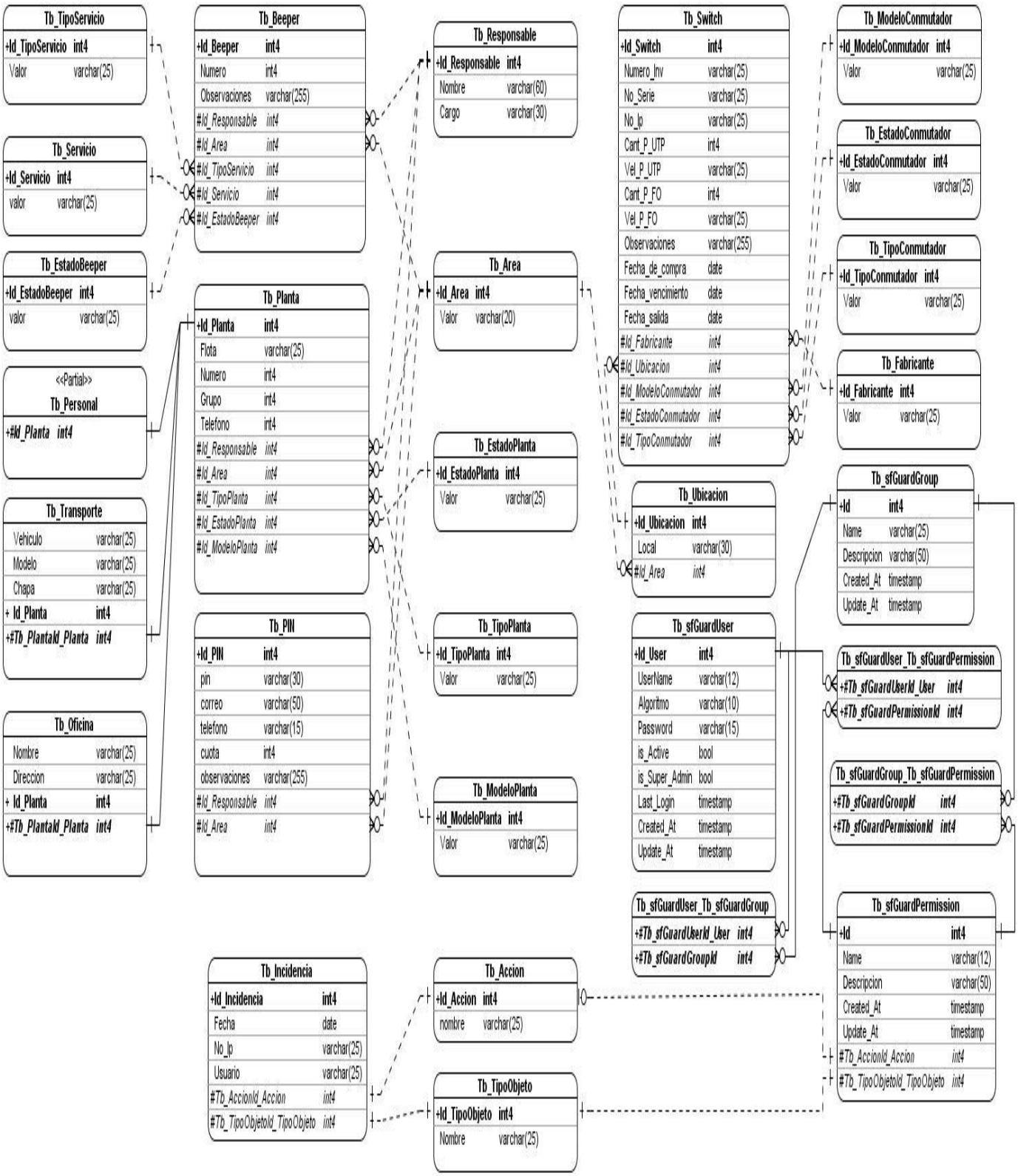


Figura 6. Diagrama entidad relación

2.11.1 Descripción de las tablas de la base de datos

En este epígrafe se define la descripción de la base de datos en forma de tablas las cuales están compuestas por el nombre de la tabla, una descripción breve de la tabla, los atributos que la componen; así como el tipo de atributo y su descripción. Solo se muestra la descripción de la tabla beeper, el resto se puede encontrar en los anexos del documento.

Nombre de la tabla: Tb_Beeper.		
Descripción: en esta tabla persistirán los datos de los beepers con que se interactúa en el negocio.		
Atributo	Tipo	Descripción
+Id_Beeper	Int4	Identificador (autoincrementado) del beeper.
Numero	Int4	Número del beeper.
Observaciones	Text	Algunas observaciones de importancia que sean válidas a tener en cuenta.
#Id_Responsable	Int4	Identificador del responsable al que pertenece el beeper.
#Id_Area	Int4	Identificador del área a la que pertenece el beeper.
#Id_TipoServicio	Int4	Identificador del tipo de servicio que posee el beeper.
#Id_Servicio	Int4	Identificador del servicio que posee el beeper.
#Id_EstadoBeeper	Int4	Identificador del estado que posee el beeper.

Tabla 17. Descripción de la tabla beeper

2.12 Modelo de despliegue

Un diagrama de despliegue muestra cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos, y la construcción interna puede ser representada por nodos o artefactos embebidos. Cómo los artefactos se ubican en los nodos

para modelar el despliegue del sistema, la ubicación es guiada por el uso de las especificaciones de despliegue. (34)

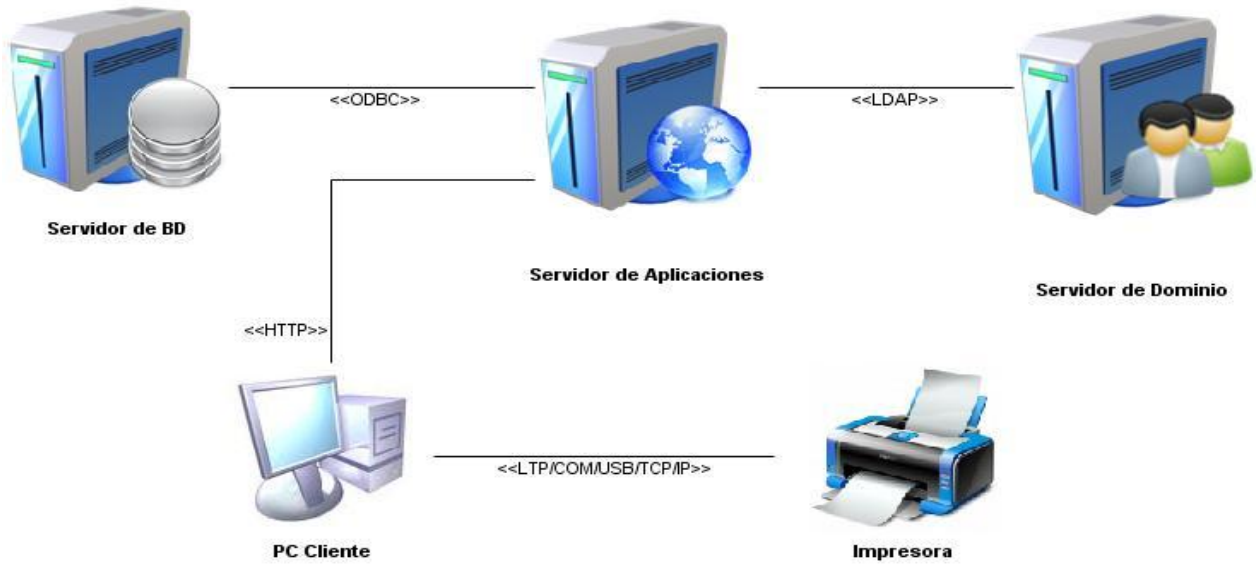


Figura 7. Diagrama de despliegue

2.13 Conclusiones del capítulo

Se obtiene una panorámica sobre las funcionalidades que el sistema debe cumplir mediante la construcción de un modelo global del sistema. Se construye una lista veintinueve funcionalidades las cuales se planean para terminar la aplicación a mediados del mes de mayo. Se identificaron un conjunto de rasgos no funcionales. Se define un patrón clásico del diseño web conocido como arquitectura Modelo Vista Controlador (MVC) para la realización del sistema. Se elaboraron los diagramas de clases del diseño, el diagrama de la base de datos, la descripción de las tablas de la base de datos y se elaboró una descripción de las principales clases del diseño para un mejor entendimiento a la hora de implementar el sistema.

CAPÍTULO III: Implementación y prueba

3.1 Introducción

Las fases implementación y prueba son las últimas del desarrollo de un sistema. Una implementación es la realización de una especificación técnica o algoritmos como un programa, componente software, u otro sistema de cómputo. Durante el proceso de implementación y prueba se deben poner en práctica todas las estrategias posibles para garantizar que el usuario inicial del sistema se encuentre libre de problemas. Un producto de software no puede alcanzar la calidad requerida sin pasar primero por un proceso exhaustivo de prueba y refinamiento. En este capítulo se aborda lo relacionado con los estándares de codificación usados para la implementación de la propuesta de solución; así como su diagrama de componentes, además se plantean las herramientas y procesos empleados para la validación de los subsistemas con la finalidad de garantizar un buen funcionamiento.

3.2 Diagrama de componente

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

A continuación se presentan los diagramas de componentes pertenecientes a la aplicación desarrollada.

Diagrama de componentes general:

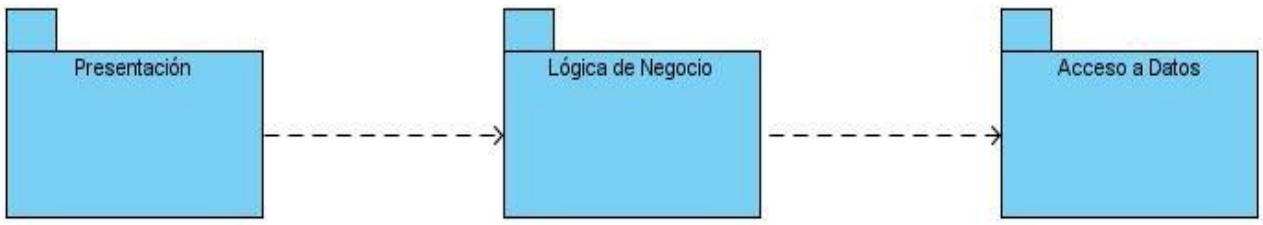


Figura 8. Diagrama de compontes general

3.3 Estándares de codificación

Es prudente establecer estándares de codificación para todos los programadores ya que estos estándares consisten en estilos de codificación a la hora de escribir el código. Los aspectos para los que generalmente se establecen estándares son los siguientes:

- Identificadores.
- Indentación.
- Líneas y espacios en blanco.
- Comentarios.

En cada grupo de desarrollo se definen cuáles serán los aspectos a estandarizar y qué estilos se aplicarán a cada uno de ellos. El cumplimiento de estándares hace que todo el código lleve el sello personal del programador y en caso de ser varios los programadores pues se busca que todo el código parezca que ha sido implementado por la misma persona. De esta manera se consigue mayor legibilidad y facilidad de mantenimiento. Los estándares deben responder además a acciones prácticas que acomoden al programador. A continuación se definirá qué estándares usar para la actividad de implementación correspondiente a este trabajo. Cabe destacar que estos estándares se definen teniendo en cuenta el estilo personal del programador, las características propias del lenguaje de programación, los recursos que se utilizarán y el tipo de programa que se debe implementar.

3.3.1 Identificadores

En el caso de los identificadores existen estilos definidos mundialmente como el *lowerCamelCase*[6] y el *UperCamelCase*. Cada palabra interna en identificadores compuestos comienza con mayúsculas para ambos estilos, además ocurre que no se colocan caracteres de separación entre las palabras que conforman un identificador compuesto en ninguno de los dos casos. Para el primero, el identificador comienza con minúscula y para el segundo, el identificador comienza con mayúscula.

[6] *CamelCase:* es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre *CamelCase* se podría traducir como Mayúsculas/Minúsculas Camello, aunque no es correcto en todos los contextos ya que la palabra inglesa Case no tiene traducción literal. El nombre se debe a que las mayúsculas a lo largo de una palabra en *CamelCase* se asemejan a las jorobas de un camello.

Clase: Para las clases se establece *UperCamelCase*.

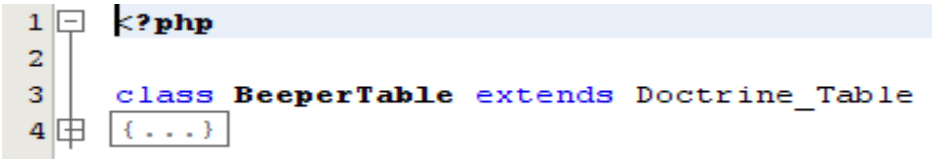


Figura 9. Ejemplo de clase que usa *UperCamelCase*

Variable: Para las variables se establece *lowerCamelCase.*

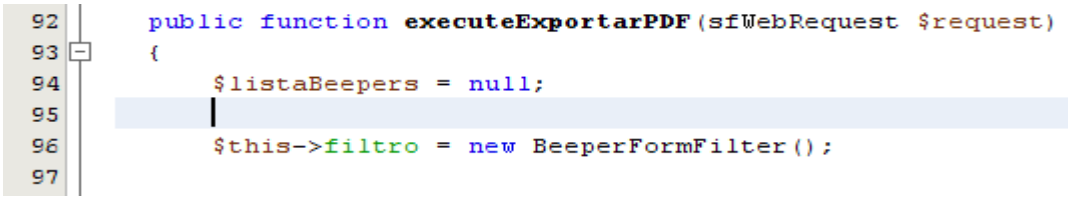


Figura 10. Ejemplo de variable que usa *lowerCamelCase*

Función: Para las funciones se establece *lowerCamelCase.*

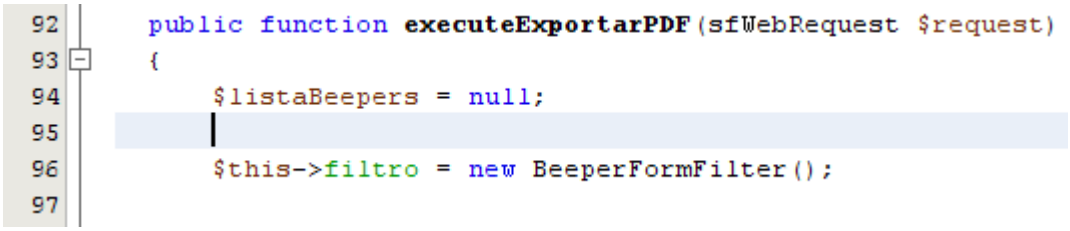


Figura 11. Ejemplo de función que usa *lowerCamelCase*

Los identificadores que sean empleados sin importar su tipo, deben ser lo suficientemente descriptivos. Con esto se evita escribir abreviaturas que pueden confundir a otros programadores por desconocer su significado. Además se deben utilizar palabras que no den un significado ambiguo.

3.3.2 Identación

La Identación es una práctica de programación que consiste en comenzar a escribir cada línea de código a diferentes distancias desde el borde izquierdo del área de texto del editor. Esta distancia está determinada por la jerarquía que se forma al introducir sentencias dentro de bloques de estructuras. Esto brinda mayor legibilidad y entendimiento para el programador e igualmente depende del propio estilo de cada persona, del lenguaje y tipo de programa que se implementa. Se definió que la Identación se hará agregando cuatro espacios al inicio de la

línea que se desee escribir. Se escribirá solo una sentencia por línea de código y en el caso de cortar las líneas, se hará luego de una coma o antes de un operador. La sección de la derecha de la línea que se corte se ubicará en la línea siguiente identada al nivel de la expresión correspondiente en la línea superior.

```
92      public function executeExportarPDF(sfWebRequest $request)
93      {
94          $listaBeepers = null;
95
96          $this->filtro = new BeeperFormFilter();
97
```

Figura 12. Ejemplo de código identado

3.3.3 Llaves

Existe diversidad de criterios en cuanto a la ubicación de las llaves que delimitan el cuerpo de los bloques de código en los lenguajes que contienen este tipo de estructuras. Algunos programadores prefieren hacerlo ubicando la llave de apertura inmediatamente detrás de la línea cabecera del bloque mientras otros apuestan por ubicarlas de forma solitaria en la línea siguiente a la línea cabecera. Para este último estilo existen además diferencias en cuanto al nivel de Identación de las mismas. Algunos lo hacen al nivel de la línea cabecera y otros al nivel de las líneas del cuerpo del bloque.

Para este trabajo:

- Las llaves de apertura se colocarán solitarias en la línea siguiente e identadas al nivel de la línea cabecera del bloque.
- Las llaves de cierre se colocarán solitarias en la línea que sigue a la última línea dentro del bloque e indentadas al nivel de la línea cabecera del bloque.

Es prudente señalar que este estilo agrega más líneas de código al programa al ubicar las llaves solitarias en una línea pero a su vez se gana en legibilidad del código.

```
92    public function executeExportarPDF(sfWebRequest $request)
93    {
94        $listaBeepers = null;
95
96        $this->filtro = new BeeperFormFilter();
97
```

Figura 13. Ejemplo de fragmento de código con estándar de llaves

3.3.4 Líneas y espacios en blanco

Para mejorar la legibilidad y organización del código muchas veces se utilizan líneas en blanco para separar segmentos de código que pueden corresponder a clases, funciones, declaraciones, implementaciones, comentarios, bloques o sencillamente secciones críticas que se deseen despejar. Así mismo, sucede con los espacios en blanco cuando se utilizan para separar elementos dentro de las sentencias de código. En ocasiones se separan con espacios cada operador de su respectivo operando, paréntesis, identificadores, símbolos y algunos lenguajes exigen que se separen las palabras propias del vocabulario de las adyacentes para ser comprendidas por los compiladores. En este trabajo se ha definido emplear **líneas en blanco**:

- Entre Funciones.

```
27    public function executeShow(sfWebRequest $request)
28    {...}
31
32    public function executeNew(sfWebRequest $request)
33    {...}
36
```

Figura 14. Líneas en blanco entre funciones

- Entre declaraciones de variables e implementaciones dentro del cuerpo de las funciones.

```
92    public function executeExportarPDF(sfWebRequest $request)
93    {
94        $listaBeepers = null;
95
96        $this->filtro = new BeeperFormFilter();
97
```

Figura 15. Líneas en blanco entre declaraciones de variables e implementaciones dentro del cuerpo de las funciones

Se colocarán **espacios en blanco:**

- Entre las palabras reservadas y los elementos adyacentes a las mismas.

```
3  class BeeperTable extends Doctrine_Table
4  {
5      public function getbeepersPaginadas($pagina = 1, $max = 10, $query = null )
6      {
7          $pager = new sfDoctrinePager('beeper', $max);
8
```

Figura 16. Espacios en blanco entre las palabras reservadas y los elementos adyacentes a las mismas

- Después de las comas en la lista de argumentos de las funciones.

```
3  class BeeperTable extends Doctrine_Table
4  {
5      public function getbeepersPaginadas($pagina = 1, $max = 10, $query = null )
6      {
7          $pager = new sfDoctrinePager('beeper', $max);
8
```

Figura 17. Espacios en blanco después de las comas en la lista de argumentos de las funciones

3.3.5 Comentarios

El uso de comentarios durante la codificación ha demostrado que es beneficiosa por varias razones:

- Ayuda al programador a entender cada elemento o sección de código.
- Hace más fácil el proceso de adaptación del código durante su reutilización.
- Sirve de guía en los casos en que varios programadores trabajen sobre las mismas secciones del código.
- Disminuye el esfuerzo de análisis ya que el lenguaje natural es más legible que cualquier lenguaje de programación.

Todo esto se aprecia claramente cuando se escribe gran cantidad de código en largos intervalos de tiempo donde generalmente el o los programadores olvidan lo que se pensó en un momento. Los comentarios se pueden utilizar para varios fines:

- Para explicar el propósito de las funciones.
- Para explicar las características fundamentales de las clases.

- Para sintetizar las acciones de los algoritmos complejos.
- Para aclarar los datos que representan las variables.
- Para dividir secciones de código en dependencia de los diferentes contextos y funciones.
- A veces se usan comentarios temporales para recordar cosas que faltan, cosas que se deben modificar o analizar en otro momento.

Es necesario tener en cuenta algunos detalles al escribir comentarios:

- La capacidad de síntesis.
- El uso de lenguaje técnico.
- No repetir exactamente paso por paso lo que hace el algoritmo, sino expresar un resumen de su propósito.
- Usar un estilo uniforme de comentario definido en estándares para todo el equipo.
- Escribir el comentario solo donde sea necesario.

En este trabajo se decidió colocar los comentarios encima de la línea a la que se le quiera aplicar y encima de la línea cabecera de los bloques. La Identación se hará al nivel de la línea en cuestión. Se utilizarán en funciones y clases. Se puede utilizar en algoritmos no triviales y secciones de diferentes contextos dentro de los métodos.

```
112         // set default header data
113         $pdf->SetHeaderData('Logouci.jpg', 50,'','Dirección de Gestión Tecnológica');
114
```

Figura 18. Ejemplo de comentario

3.4 Pruebas de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software. Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son

los válidos o esperados y los no válidos o no esperados por el programa. Además establecen los resultados a alcanzar en correspondencia de la lógica del programa y los datos ingresados. Describen las condiciones generales en las que se debe aplicar las pruebas para obtener los objetivos propuestos. El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos. Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo. Se deben planificar correctamente desde el inicio y establecer qué hacer, cómo hacer, quién va a hacer y en qué condiciones hacer las comprobaciones. Es beneficioso que los desarrolladores prueben su producto pero que no falte la mano de terceras personas que no intervinieron en el proyecto directamente ya que así se detecta mayor cantidad de fallas. (35).

Se deben escoger los tipos de prueba que se adapten mejor al sistema que se va a probar. Para esto se debe tener en cuenta el lenguaje de programación, el proceso de desarrollo, las características de los desarrolladores, el tipo de funcionalidad que se implementa, la plataforma en que se ejecutan los procesos, los errores más importantes, si la aplicación es de escritorio o web, si realiza conexiones a bases de datos, entre otras observaciones.

3.5 Automatización de pruebas

En el desarrollo de aplicaciones web el proceso de probar la aplicación de una manera correcta supone un gran esfuerzo. Esto está dado por el hecho de que los requisitos de la aplicación están sujetos a cambios constantes, lo que implica un gran número de versiones de la aplicación y con ello la aparición de nuevos errores.

Este es el motivo por el que la automatización de pruebas es una recomendación, aunque no una obligación, útil para crear un entorno de desarrollo satisfactorio. Las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación. Además, este tipo de pruebas obligan a los programadores a crear pruebas en un formato estandarizado y muy rígido que pueda ser procesado por un marco de trabajo de pruebas. (30)

El tener el proceso de prueba automatizado permite ilustrar el funcionamiento de la aplicación y al realizar un gran número de estas. Se puede mostrar la salida que produce la aplicación

para una serie de entradas de prueba, brindando la posibilidad de entender el propósito de cada método. El proceso de automatización de pruebas es la parte del ciclo de calidad en la que el software de automatización es utilizado para controlar la ejecución de pruebas, comparación de resultados, preparación de precondiciones y realización de informes. Los dos grandes grupos de pruebas unitarias existentes son las pruebas de Caja Negra y las pruebas de Caja Blanca.

3.5.1 Pruebas de caja blanca

Las pruebas de Caja Blanca se nombran de esta forma porque a diferencia de las pruebas de Caja Negra, que actúan sobre la interfaz, estas revisan la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles. Es por esto que las pruebas unitarias se basan en las Técnicas de Pruebas de Caja Blanca. Existen varios métodos que analizan diferentes partes del programa y se complementan entre sí para garantizar la calidad del sistema. La técnica del camino básico se utiliza para comprobar la complejidad lógica de un diseño procedimental, permite diseñar casos de prueba para cubrir todas las sentencias de un programa a partir de la obtención de un conjunto de caminos independientes. La complejidad ciclomática, como resultado fundamental de estas pruebas, acota la cantidad mínima de casos de prueba que se deben ejecutar. La prueba de las Condiciones es un método que se encamina hacia la ejercitación de las condiciones. Se basa en el principio de que si un conjunto de casos de prueba es capaz de ejercitar todas las condiciones contenidas en un bloque de código, este mismo conjunto serviría para encontrar más errores en el programa que no tenga que ver directamente con las condiciones. La prueba del Flujo de Datos verifica la validez en el uso de las variables para manipular los datos de la aplicación. Selecciona los casos de prueba atendiendo a las definiciones y los usos de las variables. El procedimiento indica que se debe encontrar las sentencias donde se define cada variable y las sentencias donde se hace uso de las mismas. La prueba de los Bucles se centra en la validez de las estructuras cíclicas o bucles. El objetivo es probar el comportamiento de estas estructuras en sus valores límites de iteración. Los bucles se clasifican en cuatro tipos: Bucles simples,

Bucles anidados, Bucles concatenados y Bucles no estructurados. Aunque la esencia es la misma, cada tipo se prueba de forma diferente. De lo anterior pudiera pensarse que las pruebas de Caja Blanca logran enmendar todos los errores del programa. La desventaja de estas es entonces de tipo logística ya que resulta imposible abarcar todo el código fuente de un sistema medianamente grande. El tiempo necesario para realizarlas sería considerable y se torna compleja su aplicación sobre algoritmos críticos.

3.5.2 Pruebas de caja negra

Las Pruebas de Caja Negra deben su nombre a los elementos que estas revisan y las condiciones en que se hace la revisión. Estas se basan en los requerimientos funcionales del sistema y se llevan a cabo desde el exterior de la aplicación. Este tipo de prueba es importante a la hora de medir el grado de cumplimiento de los requerimientos solicitados por el cliente y se aplican sobre la interfaz de la aplicación observando las respuestas del sistema antes determinadas acciones y los datos de salida para determinados datos de entrada. (35)

3.5.3 Pruebas unitarias y funcionales

Las pruebas unitarias aseguran que un único componente de la aplicación produce una salida correcta para una determinada entrada. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. Las pruebas unitarias se encargan de un único caso cada vez, lo que significa que un único método puede necesitar varias pruebas unitarias si su funcionamiento varía en función del contexto.

Las pruebas funcionales no solo validan la transformación de una entrada en una salida, sino que validan una característica completa. Un sistema de cache por ejemplo solamente puede ser validado por una prueba funcional, ya que comprende más de un solo paso: la primera vez que se solicita una página, se produce su código; la segunda vez, se obtiene directamente de la cache. De modo que las pruebas funcionales validan procesos y requieren de un escenario.

Pruebas Unitarias

Las pruebas unitarias permiten probar, como su nombre lo indica, cada unidad independiente del software. Actúan esencialmente sobre el código fuente y sobre los elementos básicos de la

interfaz de cada módulo. Pressman plantea que los casos de prueba que se generan durante las pruebas de unidad deben estar encaminados a verificar los siguientes elementos:

- Interfaz: “Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada”.
- Estructuras de datos locales: “Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente, conservan su integridad durante todos los pasos de ejecución del algoritmo”.
- Condiciones límites: “Se prueban las condiciones límites para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento”.
- Caminos independientes: “Se ejercitan todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez”.
- Camino de manejo de errores: “Se prueban todos los caminos de manejo de errores”.

Con las pruebas unitarias es posible aislar una parte del código de manera que pueda ser analizado. Un ejemplo de esto es evaluar las funciones o métodos, a los cuales se les realiza una entrada de datos para obtener los datos de salida correctos. Este tipo de pruebas valida la forma en la que las funciones y métodos trabajan en cada caso particular.

Pruebas Funcionales

El objetivo de las pruebas funcionales es validar si el comportamiento observado del software cumple o no con sus especificaciones. La prueba funcional toma el punto de vista del usuario. (36)

Las funciones son probadas ingresando las entradas y examinando las salidas. La estructura interna del programa raramente es considerada. (37).

Para realizar pruebas funcionales, la especificación se analiza para derivar los casos de prueba. Técnicas como partición de equivalencia, análisis del valor límite, grafo causa-efecto y conjetura de errores son especialmente pertinentes para las pruebas funcionales. Se deben considerar condiciones inválidas e inesperadas de la entrada y tener en cuenta que la definición del resultado esperado es una parte vital de un caso de la prueba. El propósito de la

prueba funcional es mostrar discrepancias con la especificación y no demostrar que el programa cumple su especificación. (38).

3.6 Propuesta de prueba del marco de trabajo Symfony

En el ámbito de PHP existen muchos marcos de trabajo para crear pruebas unitarias, siendo los más conocidos *PhpUnit* y *SimpleTest*. Symfony incluye su propio marco de trabajo llamado Lime. Se basa en la librería *Test::More* de Perl y es compatible con TAP, lo que significa que los resultados de las pruebas se muestran con el formato definido en el "*Test Anything Protocol*", creado para facilitar la lectura de los resultados de las pruebas. Lime proporciona el soporte para las pruebas unitarias, es más eficiente que otros marcos de trabajos de pruebas de PHP y tiene las siguientes ventajas:

- Ejecuta los archivos de prueba en un entorno independiente para evitar interferencias entre las diferentes pruebas. No todos los marcos de trabajos de pruebas garantizan un entorno de ejecución "limpio" para cada prueba.
- Las pruebas de Lime son fáciles de leer y sus resultados también lo son. En los sistemas operativos que lo soportan, los resultados de Lime utilizan diferentes colores para mostrar de forma clara la información más importante. Symfony utiliza Lime para sus propias pruebas, por lo que el código fuente de Symfony incluye muchos ejemplos reales de pruebas unitarias y funcionales.
- El núcleo de Lime se valida mediante pruebas unitarias.
- Está escrito con PHP, es muy rápido y está bien diseñado internamente. Consta únicamente de un archivo, llamado lime.php y no tiene ninguna dependencia. Las pruebas que se muestran en las secciones siguientes utilizan la sintaxis de Lime, por lo que funcionan directamente en cualquier instalación de Symfony. (30)

3.7 Pruebas realizadas

A continuación se muestran una serie de pruebas realizadas a los módulos más importantes que conforman la aplicación con el objetivo de validar la calidad del sistema; así como el cumplimiento de todas sus funcionalidades.

Unitarias

Las pruebas unitarias de Symfony son archivos PHP normales cuyo nombre termina en Test.php y que se encuentran en el directorio test/unit/ de la aplicación. Su sintaxis es sencilla y fácil de leer.

Para ejecutar el conjunto de pruebas, se utiliza la tarea *test:unit* desde la línea de comandos. El resultado de esta tarea en la línea de comandos es muy explícito, lo que permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente. (30)

A continuación se muestra las pruebas al módulo conmutador, las demás pruebas realizadas a los restantes módulos se encuentran anexadas en el documento.

Prueba unitaria para el módulo conmutador:

```
<?php
include(dirname(__FILE__).'/../../bootstrap/Doctrine.php');
$t = new lime_test(4);
$t->comment('Pruebas unitarias de la clase Conmutador');
$conmutador = crearConmutador() ;
$t->comment('Salvando el conmutador creado con la funcion crearConmutador()');
$cant = Doctrine::getTable('Conmutador')->count();
$t->comment(sprintf("Actualmente con %s conmutadores", $cant));
$conmutador->save();
/*************************************************************
   Pruebas
*************************************************/
$t->is($cant+1, Doctrine::getTable('Conmutador')->count(), 'Aumenta la cantidad de
conmutadores en 1');
$t->is($conmutador->getId(), !null, 'El identificador no es nulo');
$t->is($conmutador->getUbicacion()->getId(), Doctrine::getTable('Ubicacion')->find(1)-
>getId(), 'Las ubicaciones coinciden');
$t->is($conmutador->getTipoconmutador()->getId(), Doctrine::getTable('Tipoconmutador')-
>find(1)->getId(), 'Los tipos coinciden');
function crearConmutador(){
$modelo = Doctrine::getTable('Modeloconmutador')->find(1);
$ubicacion = Doctrine::getTable('Ubicacion')->find(1);
$tipo = Doctrine::getTable('Tipoconmutador')->find(1);$estado =
Doctrine::getTable('Estadoconmutador')->find(1);
$fabricante = Doctrine::getTable('Fabricante')->find(1);
```

```
    $ip = '10.33.12.226';
    $puertos = 40;
$values = array(
'numero_inventario'=> 222222,
'numero_serie'=> 212121,
'modeloconmutador_id'=> $modelo->getId(),
'estadoconmutador_id'=> $estado->getId(),
'tipoconmutador_id'=> $tipo->getId(),
'fabricante_id'=> $fabricante->getId(),
'direccion_IP'=> $ip,
'cant_Puertos_UTP'=> $puertos,
'vel_puertos_UTP'=> 40,
'cant_puertos_FO'=> 32,
'vel_puertos_FO'=> 21,
'observaciones'=> 'Esta es una prueba unitaria',
'fecha_compra'=> date('Y-M-d'),
'fecha_vencimiento'=> date('Y-M-d'),
'fecha_salida'=>date('Y-M-d'),
'ubicacion_id'=> $ubicacion->getId(),
);
$conmutador = new Conmutador();
$conmutador->fromArray($values);
return $conmutador;}
```

Resultado para la prueba unitaria del módulo conmutador:

```
1..4
# Pruebas unitarias de la clase Conmutador
# Salvando el conmutador creado con la función crearConmutador ()
# Actualmente con 4 conmutadores
ok 1- Aumenta la cantidad de conmutadores en 1
ok 2- El identificador no es nulo
ok 3- Las ubicaciones coinciden
ok 4- Los tipos coinciden
# Looks like everything went fine.
```

Con los resultados de esta prueba unitaria se comprobó que la acción *create* del módulo conmutador, la cual consiste en insertar un nuevo conmutador en la base de datos funciona correctamente y que se conserva la integridad de los datos temporales.

Funcionales

En el marco de desarrollo Symfony, las pruebas funcionales se encuentran en el directorio *test/functional/.* Las pruebas funcionales se podrían realizar mediante un navegador en forma de texto y un montón de asertos definidos con expresiones regulares complejas, pero sería una pérdida de tiempo muy grande. Symfony dispone de un objeto especial, llamado *sfBrowser*, que actúa como un navegador que está accediendo a una aplicación Symfony, pero sin necesidad de utilizar un servidor web real. Este objeto permite el acceso directo a los objetos que forman cada petición (el objeto petición, el objeto sesión, el objeto contexto y el objeto respuesta). Symfony también dispone de una extensión de esta clase llamada *sfTestBrowser*, que está especialmente diseñada para las pruebas funcionales y que tiene todas las características de *sfBrowser*, además de algunos métodos muy útiles para los asertos. (30)

Las siguientes pruebas realizan una petición a la acción por defecto del módulo y comprueba el código de estado de la respuesta, el módulo y la acción calculados por el sistema de enrutamiento y la presencia de una frase específica en el contenido de la respuesta.

A continuación se muestra las pruebas al módulo pin, las demás pruebas realizadas a los restantes módulos se encuentran anexadas en el documento.

Prueba funcional para el módulo pin:

```
<?php
include(dirname(__FILE__).'/../../bootstrap/functional.php');
$browser = new sfTestFunctional(new sfBrowser());
$browser->
 get('/')->
 click('Entrar', array(
 'signin' => array('username' => 'ysalin', 'password' => 'angelguevara123'),
 array('_with_csrf' => true)
 ))->
 with('response')->isRedirected()->
 followRedirect()->
 get('/pin/index')->
 with('request')->begin()->
 isParameter('module', 'pin')->
 isParameter('action', 'index')->
```

```
 end()->
 with('response')->begin()->
 isStatusCode(401)->
 checkElement('body', '/Acceso Denegado/')->
 end()
;
```

Resultado de la prueba funcional para el módulo pin:

```
# get /
# post /login
ok 1- page redirected to http://localhost/index.php/
# get /pin/index
ok 2- request parameter module is pin
ok 3- request parameter action is index
not ok 4- status code is 401
# Failed test (.\lib\vendor\symfony\lib\test\sfTesterResponse.class.php at line 398)
# got: 403
# expected: 401
ok 5- response selector body matches regex /Acceso Denegado/
1…5
# Looks like you failed 1 tests of 5.
```

Con los resultados de esta prueba funcional se comprobó que el usuario se autentica correctamente. Además el redireccionamiento se ejecuta de manera correcta, mostrando el error de acceso denegado por la aplicación, debido a que el usuario no posee los permisos necesarios para ejecutar la acción *index* del módulo pin. Obteniéndose resultados satisfactorios, pues el error de acceso denegado era el resultado esperado en esta prueba.

3.8 Valoración de los resultados

Una vez concluida la implementación del sistema y las pruebas realizadas al mismo, como parte de la validación se obtienen un conjunto de resultados:

- El sistema cumple con la lista de funcionalidades propuesta.
- El diseño e implementación se realizó de manera correcta.

- Se da solución a los problemas existentes que fueron detectados en las entrevistas con el cliente.
- La información que se maneja en el sistema cuenta con una correcta disponibilidad, confiabilidad e integridad.
- Las pruebas realizadas al sistema y la interacción del cliente con la aplicación resultaron satisfactorias.

A partir de estos resultados obtenidos en la investigación se puede plantear que se cumplieron los objetivos propuestos en la misma.

3.9 Estimación de esfuerzo

El Modelo Constructivo de Costes o COCOMO, por su acrónimo del inglés *Constructive Cost Model,* pertenece a la categoría de modelos de subestimaciones basados en estimaciones matemáticas. Está orientado a la magnitud del producto final, midiendo el "tamaño" del proyecto, en líneas de código principalmente. Incluye tres submodelos: básico, intermedio y detallado.

A la vez, cada submodelo también se divide en **modos** que representan el tipo de proyecto, y puede ser:

- **Modo orgánico**: un pequeño grupo de programadores experimentados desarrollan software en un entorno familiar. El tamaño del software varía desde unos pocos miles de líneas (tamaño pequeño) a unas decenas de miles (medio).
- **Modo semilibre o semiencajado**: corresponde a un esquema intermedio entre el orgánico y el rígido; el grupo de desarrollo puede incluir una mezcla de personas experimentadas y no experimentadas.
- **Modo rígido o empotrado**: el proyecto tiene fuertes restricciones, que pueden estar relacionadas con la funcionalidad y/o pueden ser técnicas. El problema a resolver es único y es difícil basarse en la experiencia, puesto que puede no haberla.

De acuerdo a las características en particular de la investigación presentada se clasificó en el modo orgánico y se calcula la estimación mediante el submodelo básico.

Modelo básico: se utiliza para obtener una primera aproximación rápida del esfuerzo.

Estos valores son para las fórmulas:

MODO	a	b	c	d
Orgánico	2.40	1.05	2.50	0.38
Semilibre	3.00	1.12	2.50	0.35
Rígido	3.60	1.20	2.50	0.32

Tabla 18. Valores para el cálculo de las fórmulas del método COCOMO

- **Personas necesarias por mes para llevar adelante el proyecto (MM) = a*(KI^{b}), donde KI es un aproximado de las líneas de código de la aplicación.**

Módulos	Cantidad de líneas de código
Beeper	10623
Pin	10784
Planta	10822
Conmutador	10725
Total	42954 SLOC (Líneas de Código Fuente)
Conversión	42954/1000=42,95 KLOC (Miles de Líneas de Código Fuente).

Tabla 19. Líneas de código por módulos

MM= 2,40*(42.95^{1,05})

MM= 124,39

- **Tiempo de desarrollo del proyecto (TDEV) = c*(MM^{d})**

TDEV= 2.50*(124,39 ^{0,38})

TDEV= 15,6 horas/hombre

- **Personas necesarias para realizar el proyecto (CosteH) = MM/TDEV**

CosteH= 124,39/15,6

CosteH= 8 personas

- **Costo total del proyecto (CosteM) = CosteH * Salario medio entre los programadores y analistas.**

CosteM= 8*100

CosteM = $ 800

Análisis de la factibilidad económica:

El desarrollo de esta aplicación no supone grandes gastos, ya sea monetario o de tiempo. Además las tecnologías utilizadas para el desarrollo de las aplicaciones son basadas en software libre. Por lo que haciendo un balance del costo-beneficio de la investigación y atendiendo al resultado que esta tributa se puede concluir que es factible de realizar.

3.10 Conclusiones del capítulo

En el presente capítulo se elaboraron los diagramas de componentes del sistema para un mejor entendimiento del mismo. Se estableció un estándar de codificación con el objetivo del que el código lleve el sello personal del programador; así como proporcionarle al código legibilidad y facilidad de mantenimiento. Se diseñaron, implementaron y realizaron las pruebas pertinentes para el sistema desarrollado antes de su puesta en explotación y se hizo una estimación del esfuerzo empleado en la elaboración de dicho sistema, concluyendo que es factible su realización.

Conclusiones Generales

Luego de entrevistar al personal de la Dirección de Gestión Tecnológica de la Universidad de las Ciencias Informáticas e identificar los procesos de gestión y control de los activos de Telecomunicaciones que se realizan en dicha universidad, se establece que es necesario un sistema que gestione y controle el flujo de actividades que se realizan con estos activos.

Para ello se realizó un estudio del estado del arte enfocado a diversos sistemas de gestión de activos, los cuales sirvieron de guía y referencia para la elaboración de este trabajo, pues los analizados no se adaptan a las características y requerimientos del que se desea desarrollar. Además, se estudió la metodología; así como las herramientas y lenguajes propuestas por el cliente para desarrollar la aplicación, estableciéndose en toda la investigación las características y ventajas del su uso.

Posteriormente se identificaron las necesidades del cliente a través de los requerimientos del software. Para la correcta solución a las exigencias del cliente a través de la metodología de desarrollo utilizada, se elaboraron las listas de funcionalidades, las que fueron planeadas, diseñadas y finalmente implementadas con el fin de obtener un sistema que gestiona y controla los activos de Telecomunicaciones en la Universidad de las Ciencias Informáticas.

El sistema desarrollado es sometido a un conjunto de pruebas forzando el mismo en diversos puntos, priorizando aquellos que fueron detectados como importantes durante el diseño e implementación del sistema, para encontrar fallos y detectar defectos, con el objetivo de obtener un producto de alta calidad, finalmente logrando satisfacer las exigencias del cliente, el cual ha quedado satisfecho con el mismo luego de su interacción con él.

Recomendaciones

Aunque los objetivos propuestos fueron cumplidos, durante el transcurso del desarrollo del sistema, han surgido una serie de nuevas ideas y recomendaciones para futuras versiones basadas en la utilidad que pueda tener; por lo cual se recomienda:

- Que la Dirección de Gestión Tecnológica continúe mejorando el sistema propuesto añadiéndole nuevas funcionalidades.
- A la Dirección de Gestión Tecnológica, extender esta aplicación a las facultades regionales de la UCI, con el objetivo de seguir informatizando las mismas.

Bibliografía referenciada

1. **Española, Real Academia**. www.rae.es. [En línea] 2010. http://www.rae.es/rae.html.
2. **The British Standards Institution**. [En línea] 2010. http://www.bsigroup.com.mx/es-mx/Auditoria-y-Certificacion/Sistemas-de-Gestion/De-un-vistazo/Que-son-los-sistemas-de-gestion/.
3. **IberSoluciones.com**. [En línea] 2007. http://www.ibersoluciones.com/que-es-un-sistema-de-gesti-n-de-contenidos.html.
4. **DDS Sistemas Informáticos**. [En línea] 2009. http://www.ddsoftware.com.ar/software-gestion-comercial.html.
5. **Cazorla, Javier**. mailxmail.com. [En línea] 2009. http://www.mailxmail.com/curso-sistema-gestion-calidad-iso-9001/que-es-sistema-gestion-calidad.
6. **Portal Web Departamento Nacional de Planación**, www.dnp.gov.co/PortalWeb/. [En línea] http://www.dnp.gov.co/PortalWeb/Gobierno/ReformadelEstado/ReformasTransversales/PROGAProgramaGesti%C3%B3ndeActivos/tabid/393/Default.aspx.
7. **DSpace.** [En línea] 2009. http://www.dspace.org/about-dspace/introducing/.
8. **Morales, MSc. Marisel Sosa Porteiro y Lic. Pedro H. Cobo**. BET-SIME: La revista del empresario cubano. [En línea] 2008. http://www.betsime.disaic.cu/secciones/eco_enemar_07.htm#2.
9. **Cubana, Rodas XXI**: un producto cubano para la empresa. www.rodasxxi.cu. [En línea] 2010. http://www.rodasxxi.cu/activos%20fijos.php.
10. **semanatecnologica.fordes.co.cu**. semanatecnologica.fordes.co.cu. [En línea] [Citado el: 20 de enero de 2010.] http://semanatecnologica.fordes.co.cu/Evirtual/files/IS046.pdf.
11. **Pressman, R. (2002).** Ingeniería de Software. Un enfoque práctico.
12. **Dámaris, Amaro Calderón y Sarah**. Metodologías Ágiles. Universidad Nacional de Trujillo, Facultad de Ciencias Físicas y Matemáticas, Escuela de Informática. 2007.
13. **Roberth G. Figueroa, Camilo J. Solís y Armando A. Cabrera**. WordPress.com. WordPress.com. [En línea] 18 de junio de 2008.

http://adonisnet.wordpress.com/2008/06/18/metodologias-tradicionales-vs-metodologias-agiles/.

14.Calabria. (2003). Web Académico Athenea . Obtenido de http://athenea.ort.edu.uy/publicaciones/ingsoft/investigacion/ayudantias/metodologia_FDD.pdf

15.Arregui, M. (2004). Docstoc. Obtenido de Docstoc: http://www.docstoc.com/docs/2143861/Tutorial-de-UML

16.SlideShare.com. (2009). Obtenido de http://www.slideshare.net/guestf131a9/herramientas-case

17.www.visual-paradigm.com. (2010). Obtenido de http://www.visual-paradigm.com/product/vpuml/

18.Departamento de Lenguajes y Sistemas Informáticos, universidad de Sevilla. (s.f.). Obtenido de http://www.lsi.us.es/~javierj/investigacion_ficheros/Marco%20de%20Trabajo.pdf

19.Eguiluz, J. (2010). Symfony.es. Obtenido de http://www.symfony.es/que-es-symfony/

20.Dana, N. (24 de marzo de 2005). Java.sun.com. Obtenido de Java.sun.com: http://translate.google.com.cu/translate?hl=es&langpair=en|es&u=http://java.sun.com/developer/technicalArticles/tools/intro.html

21.www.php.net. (2010). Obtenido de www.php.net: http://www.php.net/manual/es/index.php

22.HTMLPOINT.com. (2006). Obtenido de http://www.htmlpoint.com/javascript/corso/js_02.htm

23.España, D. w. (2010). Masadelante.com. Obtenido de Masadelante.com: http://www.masadelante.com/faqs/servidor-web

24.Ciberaula. (s.f.). Obtenido de Una Introducción a APACHE: http://linux.ciberaula.com/articulo/linux_apache_intro

25.A., E. Q. (2007). PostgreSQL.org. Obtenido de PostgreSQL.org: http://www.postgresql.org.pe/articles/introduccion_a_postgresql.pdf

26.Claudia Jiménez Quintana, L. F. (2009). Análisis de Modelos de Procesos de Negocios en relación a la dimensión informática. Obtenido de Departamento Ingeniería

Informática y <Ciencias de la Computación: http://www.inf.udec.cl/~revista/ediciones/edicion9/cjimenez.pdf

27.Felsing, S. R. (2002). A practical guide to feature-driven development.

28.Arquitectura, C. #. (2010). Entorno Virtual de aprendizaje. Obtenido de http://eva.uci.cu/mod/resource/view.php?id=14075

29.Rugarcía, B. M. (12 de enero de 2004). UDLAP Universidad de las Américas Puebla. Obtenido de http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

30.Fabien Potencier, F. Z. (2008). Symfony la guía definitiva. Licencia Creative Commons Reconocimiento -No Comercial - Sin Obra Derivada 3.0.

31.J, J. G. (2000). El proceso unificado de desarrollo de software. madrid: Addison-wesley.

32.Fernando Alonso, Loïc Martínez y Fco. Javier. *Introducción a la Igeniería de oftware.* 1ª Edición**.**

33.García, L. R. (1999). Entorno Virtual de aprendizaje. Obtenido de http://eva.uci.cu/file.php/372/Bibliografia_Basica/Libro_de_BD_de_Rosa_Maria.pdf

34.7.0, G. d. (2010). Sparx Systems. Obtenido de http://www.sparxsystems.com.ar/download/ayuda/index.html?deploymentdiagram.h

35.Lamancha, B. P. (2007). Actas de Talleres de Ingeniería del Software y Bases de Datos. Obtenido de http://www.sistedes.es/TJISBD/Vol-1/No-4/articles/pris-07-perez-gpf.pdf

36.B., I. A. (2009). CalidadySoftware.com. Obtenido de http://www.calidadysoftware.com/testing/pruebas_funcionales.php

37. Falk, C. K. (1999). Testing Computer Software.

38. Myers, G. (2004). The art of sortware testing.

Bibliografía consultada

1. **Andrei, Zeev, Rasmus, Jim y otros.** Manual de PHP.
2. **Bakken, Sæther, Schmid, Stig. y Egon.** Manual de Php. PHP Documentation Group. [En línea] http://www.php.net/docs.php.
3. **Fabien Potencier, F. Z**. (2008). Symfony la guía definitiva. Licencia Creative Commons Reconocimiento -No Comercial - Sin Obra Derivada 3.0.
4. GONZÁLEZ, M. Cuba avanza en la migración al software libre. Marzo, 2009, Disponible en: http://www.atenas.cult.cu/?q=node/6500
5. HERNÁN, S. M. Diseño de una Metodología Ágil de Desarrollo de Software. Facultad de Ingeniería. Universidad de Buenos Aires, 2004.
6. libroweb.es [Consultado el: 5 de Marzo de 2009]. Disponible en: http://www.librosweb.es/symfony/capitulo1/symfony_en_pocas_palabras.html.
7. MOLPECERES, A. Procesos de Desarrollo: RUP, XP y FDD. 2002.
8. Pizarro, Pablo. 2006. Arquitectrua de Software. Arquitectrua de Software. [En línea] 23 de mayo de 2006. http://arquitectura-de-software.blogspot.com/2006/05/orm-object-relational-mapping-ii-parte.html
9. POCESO DE DESARROLLO RUP, XP, FDD. [En línea] ALBERTO MOLPECERES, 15 de Diciembre de 2002.
10. POSTGRESQL. [En línea] POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2008. http://archives.postgresql.org/pgsql-es-ayuda/2006-05/msg00932.php
11. **Pressman, R. (2002).** Ingeniería de Software. Un enfoque práctico.
12. RUMBAUGH, J. y BOOCH, I. J. G. El Lenguaje Unificado de Modelado. Manual de Referencia.
13. symfony.es [Consultado el: 5 de Marzo de 2009]. Disponible en: http://www.symfony.es/2009/01/29/comienza-el-curso-de-symfony-en-vitoria-gasteiz/.
14. W3C. Guía breve sobre Servicios Web Disponible en: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

Glosario de términos

3FN: la tercera forma normal, a menudo abreviado 3FN, es un estándar para el desarrollo de tablas de base de datos. Es el estándar para una base de datos completamente normalizado.

ActionScript: es un lenguaje de programación orientado a objetos (OOP), utilizado en especial en aplicaciones web animadas realizadas en el entorno Adobe Flash.

Ada: es un lenguaje de programación orientado a objetos y fuertemente tipado de forma estática.

Apache: servidor web HTTP, de código abierto y multiplataforma.

BSD: *Berkeley Software Distribution* (en español, Distribución de Software Berkeley) y se utiliza para identificar un sistema operativo derivado del sistema Unix.

COCOMO: Modelo Constructivo de Costes (o **COCOMO**, por su acrónimo del inglés ***CO****nstructive* ***CO****st* ***MO****del*) es un modelo matemático de base empírica utilizado para estimación de costes de software.

CORBA IDL: lenguaje de definición de interfaces (IDL) para especificar las interfaces con los servicios que los objetos ofrecerán.

Crystal: metodología de desarrollo de software, la cual presta vital importancia a las personas que componen el equipo de un proyecto, y por tanto sus puntos de estudio son: aspecto humano del equipo, tamaño de un equipo (número de componentes), comunicación entre los componentes, distintas políticas a seguir y espacio físico de trabajo.

Delphi: entorno de desarrollo de software diseñado para la programación de propósito general con énfasis en la programación visual.

DSDM (*Dynamic Systems Development Method*): es un marco de trabajo en el que pueden entrar una gran variedad de metodologías. Combina el punto de vista de las metodologías ágiles con una especificación más rigurosa de la gestión del proyecto. Es muy útil para proyectos con restricciones temporales o requerimientos cambiantes.

GPL: ***G****eneral* ***P****ublic* ***L****icense* / Licencia Pública General. Orientada principalmente a proteger la libre distribución, modificación y uso de software.

GRASP: son patrones generales de software para asignación de responsabilidades, es el acrónimo de "*General Responsibility Assignment Software Patterns*". Aunque se considera

que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

HP Labs: laboratorios de HP (o HP *Laboratories*) es el grupo de investigación avanzada y experimental para *Hewlett-Packard Company* (es una de las mayores empresas de tecnologías de la información del mundo).

HTML (Lenguaje de Marcado de Hipertexto): lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

HTTP (Protocolo de Transferencia de Hipertexto): protocolo usado en las transacciones de la Web. Define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

IP: es una etiqueta numérica que identifica, de manera lógica y jerárquica, a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP, que corresponde al nivel de red del protocolo TCP/IP.

Java: lenguaje de programación orientado a objetos, es un lenguaje independiente de la plataforma, es compilado en un *bytecode* que es interpretado desarrollado por la compañía Sun Microsystems a principios de los 90.

JDBC: es un API para trabajar con bases de datos desde Java, independientemente de la base de datos a la que se accede.

Licencias BSD: es la licencia de software otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*). Es una licencia de software libre permisiva, permite el uso del código fuente en software no libre.

logs: son archivos en los que se recogen las visitas que tienen las páginas de un sitio web.

MAC: *Media Access Control* o control de acceso al medio. Es un identificador de 48 bits que corresponde de forma única a una ethernet de red. Se conoce también como la dirección física en cuanto a identificar dispositivos de red.

Microsoft SQL Server: sistema para la gestión de base de datos, producido por Microsoft, basado en el modelo relacional.

MIT: Instituto Tecnológico de Massachussets. Es una de las principales instituciones dedicadas a la docencia y a la investigación en Estados Unidos.

MSF (*Microsoft Solution Features*): marco de trabajo que describe las mejores prácticas en términos de principios básicos, modelos conceptuales, y disciplinas. Provee las bases descriptivas desde las cuales puede derivar cualquier metodología específica. Se considera más una metodología que un marco de trabajo. Tiene como principales características el ser altamente insistente, de planificación adaptable a los cambios y enfocado a las personas.

MySQL: sistema de base de datos relacional, multihilo y multiusuario.

Objective-C: es un lenguaje de programación orientado a objetos creado como un superconjunto de C pero que implementase un modelo de objetos parecido al de Smalltalk.

Oracle: sistema de gestión de base de datos relacional. Se considera como uno de los sistemas de bases de datos más completos, destacando: soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma.

Perl: es un lenguaje de programación. Está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.

phpDocumentor: es un generador de documentación de código abierto escrito en PHP. Automáticamente analiza el código fuente PHP y produce la API de lectura y documentación del código fuente en una variedad de formatos.

Plugins: pequeño programa que proporciona alguna funcionalidad específica a otra aplicación mayor o más compleja.

PostgreSQL: sistema de gestión de base de datos relacional orientada a objetos y libre.

Python: lenguaje de programación interpretado de código abierto.

Ruby: es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Combina una sintaxis inspirada en Python, Perl con características de programación orientada a objetos similares a Smalltalk.

RUP (*Rational Unified Process*): proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP no es un sistema

con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

SCRUM: metodología para la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software. Aunque Scrum está enfocado a la gestión de procesos de desarrollo de software, puede ser utilizado en equipos de mantenimiento de software.

Smalltalk: es un lenguaje de programación que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

Subversion: es un software de sistema de control de versiones.

Unix: sistema operativo portable, flexible, potente, con entorno programable, multiusuario y multitarea, muy difundido.

VB .NET: es un lenguaje de programación orientado a objetos que se puede considerar una evolución de Visual Basic implementada sobre el *marco de* trabajo .NET.

***Walkthrough*:** es una revisión menos formal que las inspecciones. El líder se llama coordinador y es el autor, presentador, secretario. No hay *checklists* como en las inspecciones. *Walkthroughs* se utilizan más para diseños y código.

XML (*Extensible Markup Language*): metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Es una manera de definir lenguajes para diferentes necesidades. XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores.

XML Schema: es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción.

XP (*eXtreme Programming*): metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.