SHARP-PEI

GERMÁN NADER CARDOZO

ID: 298844

LUIS ÁNGEL SUAREZ RAMÍREZ

ID: 183718

JHON FREDY GUAVITA

ID: 300759

UNIVERSIDAD MINUTO DE DIOS

TECNOLOGIA EN INFORMATICA

SOACHA

2015

SHARP-PEI

GERMÁN NADER CARDOZO

LUIS ÁNGEL SUAREZ RAMÍREZ

JHON FREDY GUAVITA

SISTEMA DE INFORMACIÓN WEB PARA LA ADMINISTRACIÓN DE LA VETERINARIA SHARP-PEI

Asesor del Proyecto

Ing. JULIO JEJÉN CARRILLO

UNIVERSIDAD MINUTO DE DIOS

TECNOLOGIA EN INFORMATICA

SOACHA

2015

PAGINA DE ACEPTACION

Nota de aceptación

_____
_____
_____

_____

Presidente del jurado

_____

Jurado

_____

Jurado

Soacha (12 de junio del 2015)

AGRADECIMIENTOS

Este trabajo que tenemos como opción de grado no habría sido posible sin la ayuda directa o indirecta de muchas personas a las que agradecemos profundamente por estar presentes en las distintas etapas de su elaboración, así como en el resto de mi vida.

Le agradecemos al profesor Julio Jején Carrillo por su ayuda y guía en el desarrollo de nuestro proyecto de grado, en cuanto a todas las etapas que debe llevar el desarrollo de software.

Le agradecemos a los administradores de la veterinaria por permitirnos diseñar un sistema de información para mejorar algunos procesos que manejan en su veterinaria y también por darnos las herramientas para poder realizar este proyecto en cuanto a la información de su veterinaria.

DEDICATORIA

En primera parte este proyecto va dedicado a todos nuestros familiares quienes siempre nos han dado su apoyo y esperan que salgamos adelante y que nuestros sueños sean cumplidos, como lo es terminar nuestra carrera universitaria.

A los colaboradores de la veterinaria desde su administrador Zulay Malaver hasta los trabajadores que hacen parte de esta, por permitirnos ingresar a lo más profundo de su desarrollo del día a día en su labor de veterinarios, pero nos han colaborado de manera desinteresada para el desarrollo del software.

Sin dejar a atrás también a todos nuestros profesores Julio Jején, Violeta Suarez y coordinador Leydy Colmenares de la facultad ya que gracias a ellos obtuvimos los conceptos necesarios para desarrollar todo nuestro proyecto, y no solo por eso, si no, por su atención en nuestro proceso para velar por nuestros sueños.

Contenido

INTRODUCCION

Por medio del siguiente documento expondremos todos los aspectos que nos facilitaron el desarrollo del proyecto, veremos entre ellos los objetivos los cuales nos plantean las metas a cumplir. La misión y visión tanto de nuestro proyecto como de la veterinaria, también como aspectos importantes tendremos el marco teórico, ciclos de vida de software, metodologías, modelamiento, diseño y otros aspectos que ayudan a que el proyecto esté bien planteado y sirva para ser implementado en la empresa.

Encontraremos tanto la teoría de los conceptos como la aplicación que le dimos al proyecto, se encontraran imágenes evidenciando su uso para mejorar la calidad del software.

JUSTIFICACION

La elaboración de este documento nos permitirá demostrar todos los temas aprendidos durante la carrera y su aplicación, la cual no solo nos servirá para graduarnos, si no, para más adelante ya que así se debe manejar el proceso de desarrollo de software en cualquier sitio.

También para permitirnos evidenciar todo el proceso que se llevó durante el desarrollo del proyecto y tener una base para demostrar lo que se hizo y como se hizo, no ayudara a tener un buen orden y presentación para futuros proyectos.

MISION

Proyecto

La misión del proyecto es ser reconocidos a nivel Soacha y Bogotá con nuestro software SHARPEI, buscamos implementar nuestro producto a muchas compañías del sector de la medicina de mascotas, con la elaboración de esta herramienta informática, proporcionamos manejo eficiente y calidad de la información en tiempo real, creando oportunidades de crecimiento y desarrollo. Para las compañías llevándolas a la vanguardia de la tecnología

Empresa

Hacer posible de la mejor manera que mejore la calidad de atención de los clientes de la veterinaria, agilizando el tiempo y utilizando como medio el software desarrollado.

VISION

Proyecto

Tener uno de los principales software de medicina de mascotas, ofreciendo soluciones tecnológicas de calidad y responsabilidad, que cumplan las necesidades de los clientes, enfrentándonos a los cambios, siendo reconocidos con la confianza de las persona que requieren nuestro servicios.

Empresa

Poder ser la veterinaria preferida por la gente que vive en Soacha y poder obtener la fidelidad de esta.

OBJETIVOS

General

Analizar, diseñar e implementar un Sistema de Información Web para la administración de la veterinaria SHAR-PEI.

Específicos

✓ Crear un módulo para clientes que permita consultar, insertar, modificar y eliminar información correspondiente a cada uno de ellos.

✓ Crear una base de datos en MYSQL para el control de clientes, proveedores, productos.

✓ Desarrollar un formulario que muestre a los usuarios los servicios prestados por la veterinaria.

✓ Crear un módulo para el registro de citas.

✓ Permitir al cliente una solución vía web para agilizar la asignación de citas y evitar el gasto innecesario de tiempo.

ESTUDIO DE CAMPO

VISITA DE CAMPO: Se hacen visitas de campo a la veterinaria en dos etapas en primera instancia se hace las visitas en la semana lunes a viernes encontrando allí que no había mucha afluencia de clientes, y en segunda instancia se hace el

fin de semana comprendidos sábado y domingo observando que estos dos días son los preferidos para hacer las visitas a el veterinario.

ENCUESTAS: Mediante las encuestas se verifica lo presenciado en la visita campo, que las personas por cuestión de tiempo y facilidad hacen uso del fin de semana para dedicarle unas horas a sus mascotas

ENTREVISTA: Al igual, de la información recolectada en los anteriores estudios se asevera por parte de la administradora la necesidad de poder hacer mejor uso de sus tiempos en la semana comprendida de lunes a domingo para que su veterinaria no sea frecuentado únicamente los fines de semana sino que también sea frecuentada entre semana, esto por medio del ofrecimiento de productos y servicio médico utilizando un software que le facilite a el cliente la atención de la veterinaria.

MARCO TEORICO

Las veterinarias en Soacha no cuentan con un software que les brinden la suficiente información a los usuarios, para agendar una cita para la mascota necesariamente debe acercase al punto de atención para generar la respectiva cita esto puede ser un poco tedioso, el usuario no tiene conocimiento de la información instalaciones de la veterinaria.

Esta información se logró obtener a través de una encuesta de 100 personas con mascotas (perros y gatos) en el municipio de Soacha en los barrios San Carlos, San Mateo, Ciudad Latina y León Trece y se llega a la misma conclusión, la falta de información hacia el usuario por parte de las veterinarias en donde son atendidos sus respectivas mascotas.

MARCO HISTORICO

A nivel de desarrollo de Software para las veterinarias, y particularmente en Soacha; no existe una empresa dedicada al desarrollo de soluciones informáticas para este sector. Es así como se hace necesario dar una mirada a estas compañías de salud para mascotas, investigando e indagando en la creación de herramientas que permitan llevar el control adecuado de las diferentes actividades que las empresas de este ramo pueden desarrollar a diario, buscando hacer más eficientes sus procesos productivos y que se conviertan en mayores ingresos económicos para sus propietarios

MARCO REFERENCIAL

La veterinaria SHAR-PEI se encuentra ubicada en la Calle 18 No. 6 – 15 Soacha (Cundinamarca); actualmente en Colombia, los sistemas de información y/o de gestión son traídos de países extranjeros como Argentina, quienes tienen dos Sistemas de Gestión llamados QVET y Vetter Sistemas, teniendo costos elevados para Colombia y el resto de Latinoamérica. Pretendemos implementar un Sistema de Información desarrollado en Colombia, para que salga al mercado y facilite el manejo de controles y registros pertinentes que se deben llevar en una veterinaria.

El proyecto surge como necesidad a petición de la administración de la veterinaria SHAR-PEI, ya que notaron demasiada lentitud y baja eficacia en sus procesos manejados en archivos físicos, requieren sistematizar cada uno de estos para agilizar y mejorar la atención al cliente.

MARCO LEGAL

Es necesario tener en cuenta algunas directrices para la ejecución, venta y distribución de Software los cuales son: Derechos de Autor y propiedad intelectual, ley de habeas data y publicidad. De acuerdo a lo anterior, La Propiedad Intelectual es una disciplina normativa que protege las creaciones intelectuales provenientes de un esfuerzo, trabajo o destreza humanos, dignos de reconocimiento jurídico. La Propiedad Intelectual comprende:

El derecho de autor y los derechos conexos.

¿Qué es el derecho de autor?

Es la protección que le otorga el Estado al creador de las obras literarias o artísticas desde el momento de su creación. Merece la pena señalar, en este punto, que existen varios tipos de infracción de los derechos de autor del software; y todos ellos deben evitarse. Los tipos de infracción más comunes son los siguientes:

· Utilización sin licencia alguna; por ejemplo, copiar un programa de software de un amigo o de Internet, etc., cuando la licencia del software no lo permita explícitamente.

· Uso abusivo; por ejemplo, comprar un software licenciado para un ordenador e instalarlo en dos.

· Fallo en la transmisión de la licencia o incapacidad para volver a licenciar; cuando se adquiere hardware de segunda mano, no necesariamente se van a transmitir todas las licencias de software, por lo que debemos tomar las medidas oportunas para asegurarnos de que el uso es legítimo.

· Obtención del software de manera fraudulenta; por ejemplo, conseguir una reducción del precio aparentando que su empresa es una institución educativa.

· descargan y usan copias piratas infringen derechos de autor.

· "Ofertas especiales" ilícitas de vendedores de hardware; cuando un vendedor de hardware vende un ordenador con el software instalado, pero el software no está licenciado (lo más habitual es que el cliente no lo sepa).

· Hacer una copia ilícita del software en un CD-ROM grabable u otro soporte similar, con la intención de dársela a otra persona.

· Falsificación, es decir, la realización de copias ilegales de software en CD-ROM grabables, o soportes similares, a escala comercial y su venta como si fuesen copias legales (utilizando carátulas engañosas, etc.). La falsificación es dominio exclusivo de los delincuentes profesionales. Si el software se encuentra a un precio considerablemente reducido, bien podrá tratarse de una falsificación. y por un tiempo determinado.

METODO CLASICO DE CICLO DE VIDA

Definición: El método del Ciclo de Vida para el desarrollo de sistemas (SDLC), es el conjunto de actividades que los analistas, diseñadores y usuarios realizan para desarrollar e implantar un sistema de información.

Esta consta de las siguientes partes:

1. Investigación preliminar:

Si un proyecto de sistema parece ser viable y tiene suficiente prioridad, se comienza la investigación preliminar. Esta investigación requiere uno o más

analistas de sistemas analizando el “System Request” para determinar la verdadera naturaleza y alcance del problema y recomendar si es que se debe continuar con el proyecto.

2. Determinación de los requerimientos del sistema:

Las empresas deben entender todas las facetas correspondientes donde se realiza o se quiere implementar el desarrollo del sistema. Las personas encargadas en la elaboración y desarrollo deben estudiar los procesos de una empresa para dar respuesta a las necesidades exigidas por el cliente.

3. Diseño del Sistema

El diseño de un sistema de información comprende los detalles que establece la manera en como el sistema cumplirá con los requerimientos obtenidos en la fase de análisis.

4. Desarrollo del Software.

Hay diferentes alternativas para el desarrollo del software. La primera es la elaboración del mismo por parte de programadores familiarizados con el desarrollo del proyecto y la segunda es la instalación o modificación de un software comprado que satisfaga las necesidades del cliente que solicitud este desarrollo de sistema. Esto dependerá de los costos asociados con la implementación del mismo y del tiempo primordialmente.

5. Prueba de los sistemas.

Durante el desarrollo de esta fase se emplea el sistema de manera experimental y así determinar si no se encuentran fallas que hagan que no se respeten los requerimientos de su elaboración. Se alimentan con entradas, conjunto de datos para su procesamiento y luego se verifica los resultados.

6. Implantación y evaluación.

La implantación es el proceso de verificar e instalar nuevo equipo, entrenar los usuarios, instalar la aplicación y construir todos los archivos.

En la parte de Evaluación, se identifican puntos débiles y fuertes. La evaluación ocurre a lo largo de las diferentes dimensiones

MODELO ESPIRAL

El modelo espiral en el desarrollo del software es un modelo meta del ciclo de vida del software donde el esfuerzo del desarrollo es iterativo, tan pronto culmina un esfuerzo del desarrollo por ahí mismo comienza otro; además en cada ejecución del desarrollo se sigue cuatro pasos principales:

1. Determinar o fijar los objetivos. En este paso se definen los objetivos específicos para posteriormente identifica las limitaciones del proceso y del sistema de software, además se diseña una planificación detallada de gestión y se identifican los riesgos.

2. Análisis del riesgo. En este paso se efectúa un análisis detallado para cada uno de los riesgos identificados del proyecto, se definen los pasos a seguir para reducir los riesgos y luego del análisis de estos riesgos se planean estrategias alternativas.

3. Desarrollar, verificar y validar. En este tercer paso, después del análisis de riesgo, se eligen un paradigma para el desarrollo del sistema de software y se lo desarrolla.

4. Planificar. En este último paso es donde el proyecto se revisa y se toma la decisión si se debe continuar con un ciclo posterior al de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

Con cada iteración alrededor de la espiral, se crean sucesivas versiones del software, cada vez más completas y, al final, el sistema de software ya queda totalmente funcional.

La diferencia principal entre el modelo espiral y los modelos anteriores (ej.: cascada, evolutivo, incremental, etc.) es la evaluación del riesgo. El riesgo es todo lo que pueda salir mal en un proyecto de desarrollo de software. Por ejemplo, si queremos utilizar un lenguaje de programación para desarrollar un sistema operativo, un riesgo posible es que los compiladores utilizables no produzcan un código objeto eficiente. Los riesgos originan problemas en el proyecto, como el exceso de los costos. Es así que, la disminución de los riesgos es una actividad muy importante.

Un modelo espiral comienza con la determinación de los objetivos tanto funcionales como de rendimiento. Después se enumeran algunas formas posibles de alcanzar estos objetivos identificando las fuentes de riesgos posibles. Luego continuamos con el siguiente paso que es resolver estos riesgos y llevar a cabo las actividades de desarrollo, para finalizar con la planificación del siguiente ciclo de la espiral.

CARACTERÍSTICAS DEL MODELO EN ESPIRAL PARA EL DESARROLLO DE SOFTWARE

Es considerado como un modelo evolutivo ya que combina el modelo clásico con el diseño de prototipos.

Contiene una nueva etapa que es el análisis de riesgos, no incluida anteriormente.

Este modelo es el indicado para desarrollar software con diferentes versiones actualizadas como se hace con los programas modernos de PC´s.

La ingeniería puede desarrollarse a través del ciclo de vida clásico o el de construcción de prototipos.

Este es el enfoque más realista actualmente.

El modelo en espiral esta compartida en varias actividades estructurales, también llamadas regiones de tareas. Existen seis regiones de tareas que son:

Comunicación con el cliente: esta es una tarea requerida para establecer comunicación entre el desarrollador y el cliente.

Planificación: esta tarea es necesaria aplicarla para poder definir los recursos, el tiempo y otras informaciones relacionadas con el proyecto, es decir, son todos los requerimientos.

Análisis de riesgos: esta es una de las tareas principales por lo que se aplica el modelo en espiral, es requerida para evaluar los riesgos técnicos y otras informaciones relacionadas con el proyecto.

Ingeniería: esta es una tarea necesaria ya que se requiere construir una o más representaciones de la aplicación.

Construcción y adaptación: esta tarea es requerida en el modelo espiral porque se necesita construir, probar, instalar y proporcionar soporte al usuario.

Evaluación el cliente: esta también es una tarea principal, necesaria para adquirir la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería y la de implementación creada durante la etapa de instalación.

VENTAJAS DEL MODELO ESPIRAL

- No requiere una definición completa de los requerimientos del software a desarrollar para comenzar su funcionalidad.

- En la terminación de un producto desde el final de la primera iteración es muy factible aprobar los requisitos.

- Sufrir retrasos corre un riesgo menor, porque se comprueban los conflictos presentados tempranamente y existe la forma de poder corregirlos a tiempo.

DESVENTAJAS DEL MODELO ESPIRAL

- Existe complicación cuando se evalúa los riesgos.

- Se requiere la participación continua por parte del cliente.

- Se pierde tiempo al volver producir inicialmente una especificación completa de los requerimientos cuando se modifica o mejora el software.

ACOPLAMIENTOS DEL MODELO ESPIRAL

Los nuevos requerimientos del sistema se definen en todo los detalles posibles, esto implica generalmente el entrevistarse con un número determinado de usuarios que representarán a todos los usuarios tanto externos como internos y otros aspectos del sistema existente.

Un prototipo preliminar se crea para el desarrollo del nuevo software partiendo de un diseño hecho del sistema que se construyó del prototipo inicial. Esto es

generalmente un sistema scaled-down, y representa una aproximación de las características del producto final.

Un segundo diseño de software es desarrollado por un procedimiento cuádruple:

- Evaluación del primer prototipo en términos de sus fuerzas, debilidades, y riesgos;

- Definir los requisitos del segundo prototipo;

- Planeando y desarrollando el segundo prototipo;

- Construyendo y probando el segundo prototipo.

En la opción del cliente, el proyecto completado puede ser abortado si el riesgo se juzga demasiado grande. Los factores de riesgo pudieron implicar los excesos de coste del desarrollo, cálculo erróneo del fusionar los costes, o cualquier otro factor que podría, en el juicio del cliente, dar lugar a un producto final menos que satisfactorio.

El diseño existente se evalúa de manera semejante al igual que el diseño anterior, y, en caso de necesidad, otro prototipo se desarrolla de él según el procedimiento cuádruple expuesto anteriormente.

Se iteran los pasos precedentes hasta que el cliente está satisfecho sabiendo que el diseño mejorado representa el producto final deseado. Además, se construye el

sistema final, basado en el diseño mejorado. El sistema final se evalúa y se prueba con todas las de ley. El mantenimiento general se realiza sobre una base continua para prevenir fallas en grande y para reducir al mínimo el tiempo perdido.

CONCLUCIÓN

El prototipo del modelo en espiral para la ingeniería de software es en la actualidad el enfoque más realista para el desarrollo de software y de sistemas a gran escala. Utiliza un enfoque evolutivo para la ingeniería de software, permitiendo al desarrollador y al cliente entender y reaccionar a los riesgos en cada nivel del modelo en espiral.

Utiliza la creación de prototipos como un mecanismo de reducción de riesgo, pero, lo que es más importante permite a quien lo desarrolla aplicar el enfoque de creación de prototipos en cualquier etapa de la evolución de prototipos.

CICLO DE VIDA-MODELO CASCADA

En los años 70 se impuso un nuevo enfoque de desarrollo del software, introducido por Royce en 1970, a través de un ciclo de vida en “cascada” (así denominado por la disposición de las distintas fases de desarrollo, en las que los

resultados de una fase parecen caer en cascada hacia la siguiente fase, tal como se muestra en la Figura 1).

El método ideado por Royce constituye uno de los primeros modelos de ciclo de vida publicados, por lo que también recibe el nombre de modelo de ciclo de vida clásico. Este método modela el ciclo convencional de la Ingeniería del Software, aplicando un enfoque sistemático y secuencial de desarrollo que comienza con la ingeniería del sistema y progresa a través del análisis, diseño, codificación, pruebas y mantenimiento.

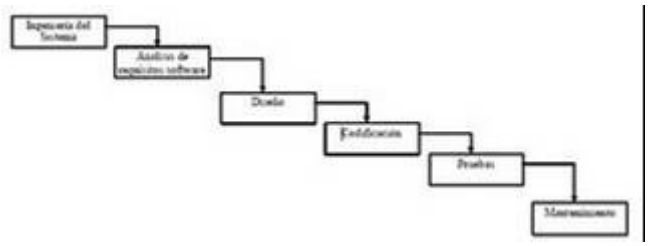


Figura 1 Ciclo de vida en cascada o clásico

Como sugiere el esquema del modelo en cascada, antes de poder avanzar a la siguiente etapa, es necesario haber finalizado completamente la etapa anterior. Asociada con cada etapa del proceso existen hitos y documentos, de tal forma que se puede utilizar el modelo para comprobar los avances del proyecto y para estimar cuánto falta para su finalización.

Este modelo es muy útil pues ayuda a los desarrolladores a comprender qué es lo que tienen que hacer en cada momento. Su simplicidad hace que resulte sencillo

explicárselo a los clientes que no están familiarizados el proceso software. Además, se muestran de forma explícita qué productos intermedios se tienen que obtener antes de abordar las siguientes tareas.

Una modificación sobre este modelo consiste en la introducción de una revisión y vuelta atrás, con el fin de corregir las deficiencias detectadas durante las distintas etapas, o para completar o aumentar las funcionalidades del sistema en desarrollo, resultando un diagrama de fases y etapas tal como el que se muestra en la Figura 2.

De esta manera, durante cualquiera de las fases se puede retroceder momentáneamente a una fase previa para solucionar los problemas que se pudieran haber encontrado.

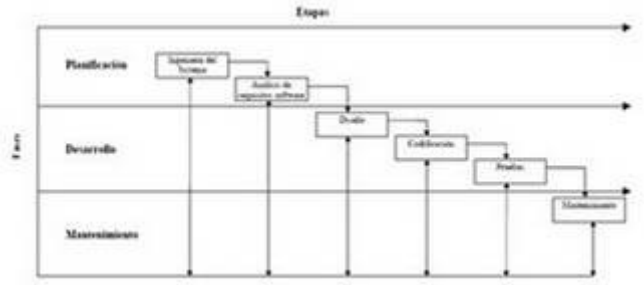


Figura 2 Ciclo de vida en cascada o clásico completo

Normalmente, el ciclo de vida del software se suele dividir en tres fases: una de Planificación, otra de Desarrollo y una tercera de Mantenimiento, que engloban a

las seis etapas (Ingeniería del Sistema, Análisis de los Requisitos, Diseño, Codificación, Pruebas y Mantenimiento) tradicionales del ciclo de vida.

La fase de Planificación del software comprende las etapas de Ingeniería del Sistema o Análisis del Sistema (en concreto el establecimiento de los Requisitos del Software o “Plan Software”) y el Análisis de los Requisitos del Software (que se traduce en una “Especificación de Requisitos”). La fase de Desarrollo comprende las etapas de Diseño, Codificación y Pruebas. Por último, la fase de Mantenimiento incorpora solamente la etapa propia de Mantenimiento.

A pesar de su antigüedad, el ciclo de vida clásico se ha hecho con un lugar importante en el área de la Ingeniería del Software. Proporciona una guía de trabajo en la que se encuentran métodos para el análisis, diseño, codificación, pruebas y mantenimiento. El ciclo de vida en cascada sigue siendo el modelo de proceso más extensamente utilizado por los ingenieros del software, principalmente por su sencillez y facilidad de llevar a cabo. Pese a tener debilidades, es significativamente mejor que un enfoque arbitrario (como el de codificar y corregir) para el desarrollo del software. Muchos de los posteriores modelos de ciclo de vida son, en realidad, modificaciones sobre el modelo clásico, al que se le incorporan iteraciones o nuevas actividades.

Ventajas y desventajas del Modelo en “cascada”

Ventajas:

• Es un modelo sencillo y disciplinado

• Es fácil aprender a utilizarlo y comprender su funcionamiento

• Está dirigido por los tipos de documentos y resultados que deben obtenerse al final de cada etapa

• Ha sido muy usado y, por tanto, está ampliamente contrastado

• Ayuda a detectar errores en las primeras etapas a bajo costo

• Ayuda a minimizar los gastos de planificación, pues se realiza sin problemas

Desventajas:

• Los proyectos raramente siguen el proceso lineal tal como se definía originalmente el ciclo de vida

• Es difícil que el cliente exponga explícitamente todos los requisitos al principio

• El cliente debe tener paciencia pues obtendrá el producto al final del ciclo de vida

• No refleja exactamente cómo se programa realmente el sistema, en el que suele haber un gran componente iterativo

• Puede resultar complicado regresar a etapas anteriores (ya acabadas) para

realizar correcciones

• El producto final obtenido puede que no refleje todos los requisitos del usuario.

CICLO DE VIDA ORIENTADO A OBJETOS

Los tipos de Modelos de ciclos de vida normalmente se basan el análisis y diseño estructurados, pero los objetos tienen una particularidad, y es que están basados en componentes que se relacionan entre ellos a través de interfaces, o lo que es lo mismo, son más modulares y por lo tanto el trabajo se puede dividir en un conjunto de mini proyectos. Además, hoy en día la tendencia es a reducir los riesgos, y en este sentido, el ciclo de vida en cascada no proporciona muchas facilidades. Debido a todo esto, el ciclo de vida típico en una metodología de diseño orientado a objetos es iterativo e incremental.

COMPONENTES DEL MODELO DE ESTRUCTURA DE OBJETOS

El componente básico del OSM es la clase de objetos. Se distinguen tres tipos de clase:

Objetos Entidad, objetos de Interfaz, objetos de Control.

CONCEPTO Y ESTRUCTURA DE EL MODELO DE ESTRUCTURA DE OBJETOS

El OSM es el modelo fundamental que provee un medio uniforme para modelar el sistema desde la captura de requerimientos en la etapa inicial del análisis hasta la complementación, atravesando todo el ciclo de desarrollo del sistema.

Este modelo identifica:

- Las clases de objetos en la aplicación.
- Como las clases de objetos se asocian unas con otras.
- Como se comunican los objetos.
- Los detalles de cada clase de objetos, incluyendo atributos y operaciones.

Durante el proceso de análisis y diseño, el OSM es definido en sucesivos niveles incrementales de detalle, hasta que el nivel necesario para la complementación es alcanzado.

Todos los demás modelos capturan detalles que alimentan es modelo.

El desarrollo de OSM es un proceso aditivo, diferenciándose esto del enfoque transformacional característico de otros métodos como el estructurado, donde los DFD del análisis son transformados en diagramas de estructura durante el diseño, con los consiguientes problemas que esto acarrea.

Durante el ciclo de desarrollo se aportan los siguientes elementos al modelo:

- Análisis del Negocio: se reconocen objetos claves del negocio y generan las abstracciones en las clases apropiadas (objetos entidad).

- Análisis de Requerimientos: se identifican asociaciones estructurales entre objetos y nuevas clases (entidad).

- Diseño lógico: Se incorporan todas las clases necesarias para la aplicación incluyendo los objetos de interfaz y de control.

- Diseño Físico: se incorporan todos los detalles remanentes para la complementación física de cada clase de objetos.

VENTAJAS

- La planificación es sencilla.

- La calidad del producto resultante es alta.

- Permite trabajar con personal poco cualificado

DESVENTAJAS

- Lo peor es la necesidad de tener todos los requisitos al principio. Lo normal es que el cliente no tenga perfectamente definidas las especificaciones del sistema, o puede ser que surjan necesidades imprevistas.

- Si se comete un error en la fase de análisis no lo descubrimos hasta la entrega, con el consiguiente gasto inútil de recursos.

METODOLOGÍA RUP

Es una metodología cuyo fin es entregar un producto de software. Se estructura todos los procesos y se mide la eficiencia de la organización.
Es un proceso de desarrollo de software el cual utiliza el lenguaje unificado de modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.
El RUP es un conjunto de metodologías adaptables al contexto y necesidades de cada organización.
Describe como aplicar enfoques para el desarrollo del software, llevando a cabo unos pasos para su realización.
Se centra en la producción y mantenimiento de modelos del sistema.

Principales características

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).

- Pretende implementar las mejores prácticas en Ingeniería de Software.

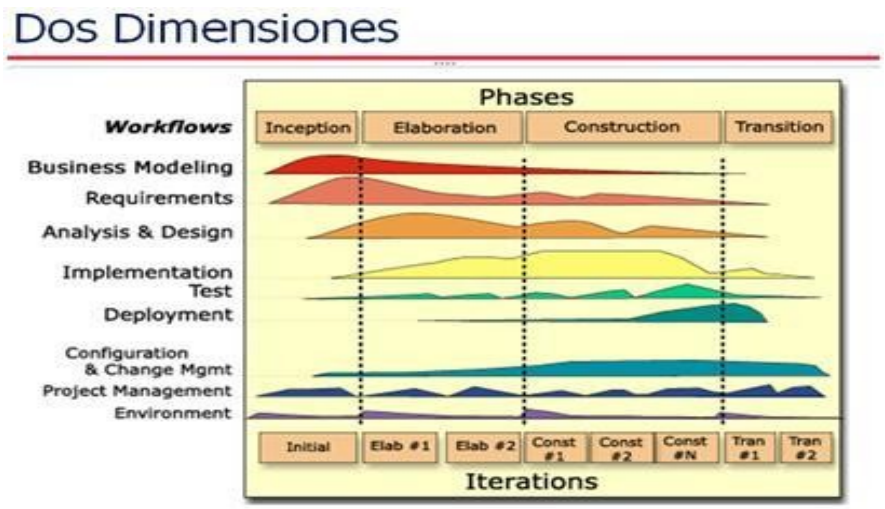
- Desarrollo iterativo.

- Administración de requisitos.

- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.
- Verificación de la calidad del software.

El RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

CICLO DE VIDA



Esfuerzo en actividades según fase del proyecto

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

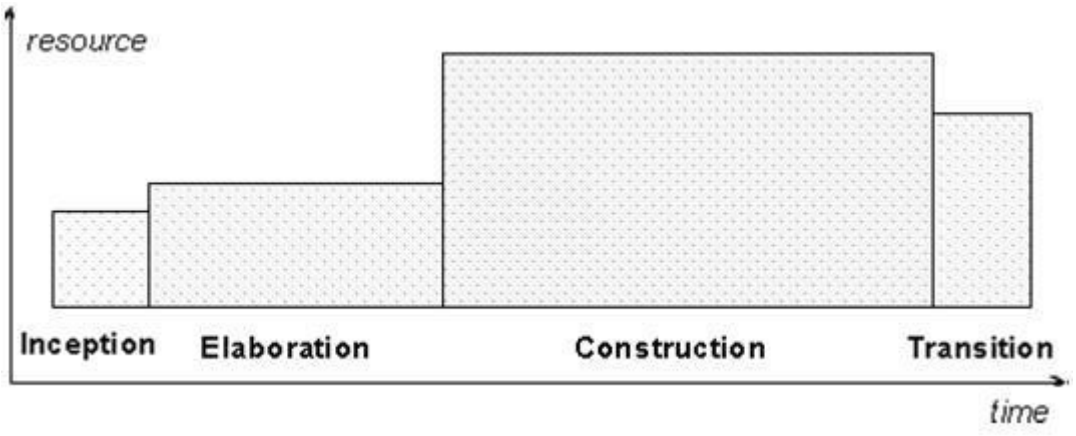
Fases del ciclo de vida del RUP:

1. Fase de Inicio: Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones posteriores.

2. Fase de elaboración: En la fase de elaboración se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar.

3. Fase de Desarrollo: El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y se realizan las mejoras para el proyecto.

4. Fase de Cierre: El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto.



La metodología RUP tiene 6 principios clave:

1. Adaptación del proceso: El proceso debe adaptarse a las características de la organización para la que se está desarrollando el software.

2. Balancear prioridades: Debe encontrarse un balance que satisfaga a todos los inversores del proyecto.

3. Colaboración entre equipos: Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, entre otros.
4. Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de una forma interna, en etapas iteradas. En cada iteración se evaluará la calidad y estabilidad del producto y analizará la opinión y sugerencias de los inversores.

5. Elevar el nivel de abstracción: Motivar el uso de conceptos reutilizables.

6. Enfocarse en la calidad: La calidad del producto debe verificarse en cada aspecto de la producción.

Disciplina de desarrollo de RUP

Determina las etapas a realizar durante el proyecto de creación del software.

Ingeniería o modelado del negocio: Analizar y entender las necesidades del negocio para el cual se está desarrollando el software.

- Requisitos: Proveer una base para estimar los costos y tiempo de desarrollo del sistema.

- Análisis y diseño: Trasladar los requisitos analizados anteriormente a un sistema automatizado y desarrollar una arquitectura para el sistema.

- Implementación: Crear software que se ajuste a la arquitectura diseñada y que tenga el comportamiento deseado.

- Pruebas: Asegurarse de que el comportamiento requerido es correcto y que todo lo solicitado está presente.

- Despliegue: Producir distribuciones del producto y distribuirlo a los usuarios.

Disciplina de soporte RUP

Determina la documentación que es necesaria realizar durante el proyecto.

- Configuración y administración del cambio: Guardar todas las versiones del proyecto.

- Administración del proyecto: Administrar los horarios y recursos que se deben de emplear.

- Ambiente: Administrar el ambiente de desarrollo del software.

- Distribución: Hacer todo lo necesario para la salida del proyecto.

Elementos del RUP

- Actividades: Procesos que se han de realizar en cada etapa/iteración.

- Trabajadores: Personas involucradas en cada actividad del proyecto.

- Artefactos: Herramientas empleadas para el desarrollo del proyecto. Puede ser un documento, un modelo, un elemento del modelo.

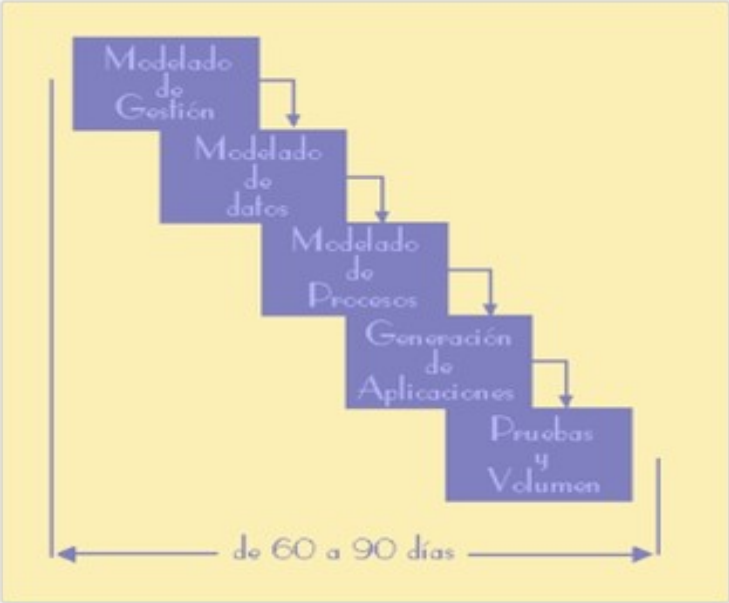
METODOLOGIA RAD

El desarrollo rápido de aplicaciones o RAD (acrónimo en inglés de rapid application development) es un proceso de desarrollo de software, desarrollado inicialmente por James Martin en1980. El método comprende el desarrollo interactivo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering). Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario tales como Glade, o entornos de desarrollo

integrado completos. Algunas de las plataformas más conocidas son Visual Studio, Lazarus, Gambas, Delphi,Foxpro , Anjuta, Game Maker, Velneo o Clarion. En el área de la autoría multimedia, software como Neosoft Neoboo y MediaChance Multimedia Builder proveen plataformas de desarrollo rápido de aplicaciones, dentro de ciertos límites.

FASES DEL RAD



- Modelado de gestión: el flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la proceso?

- Modelado de datos: el flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos

necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.

- Modelado de proceso: los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos. Es la comunicación entre los objetos.

- Generación de aplicaciones: El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.

- Pruebas de entrega: Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

¿PORQUÉ USAR RAD?

Malas razones

- Prevenir presupuestos rebasados (RAD necesita un equipo disciplinado en manejo de costos).

- Prevenir incumplimiento de fechas (RAD necesita un equipo disciplinado en manejo de tiempo).

Buenas razones

- Convergir tempranamente en un diseño aceptable para el cliente y posible para los desarrolladores.

- Limitar la exposición del proyecto a las fuerzas de cambio.

- Ahorrar tiempo de desarrollo, posiblemente a expensas de dinero o de calidad del producto.

CARACTERÍSTICAS DE RAD

Equipos Híbridos

- Equipos compuestos por alrededor de seis personas, incluyendo desarrolladores y usuarios de tiempo completo del sistema así como aquellas personas involucradas con los requisitos.

- Los desarrolladores de RAD deben ser "renacentistas": analistas, diseñadores y programadores en uno.

Herramientas Especializadas

- Desarrollo "visual"
- Creación de prototipos falsos (simulación pura)
- Creación de prototipos funcionales
- Múltiples lenguajes
- Calendario grupal
- Herramientas colaborativas y de trabajo en equipo
- Componentes reusables
- Interfaces estándares (API)
- "Timeboxing"
- Las funciones secundarias son eliminadas como sea necesario para cumplir con el calendario.

Prototipos Iterativos y Evolucionarios.

- Reunión JAD (Joint Application Development):
- Se reunen los usuarios finales y los desarrolladores.

- Lluvia de ideas para obtener un borrador inicial de los requisitos.
- Iterar hasta acabar:
- Los desarrolladores construyen y depuran el prototipo basado en los requisitos actuales.
- Los diseñadores revisan el prototipo.
- Los clientes prueban el prototipo, depuran los requisitos.
- Los clientes y desarrolladores se reúnen para revisar juntos el producto, refinar los requisitos y generar solicitudes de cambios.
- Los cambios para los que no hay tiempo no se realizan. Los requisitos secundarios se eliminan si es necesario para cumplir el calendario.

VENTAJAS

- Comprar puede ahorrar dinero en comparación con construir.
- Los entregables pueden ser fácilmente trasladados a otra plataforma.
- El desarrollo se realiza a un nivel de abstracción mayor.
- Visibilidad temprana.

- Mayor flexibilidad.
- Menor codificación manual.
- Mayor involucramiento de los usuarios.
- Posiblemente menos fallas.
- Posiblemente menor costo.
- Ciclos de desarrollo más pequeños.
- Interfaz gráfica estándar.

DESVENTAJAS

1. Comprar puede ser más caro que construir.
2. Costo de herramientas integradas y equipo necesario.
3. Progreso más difícil de medir.
4. Menos eficiente.
5. Menor precisión científica.
6. Riesgo de revertirse a las prácticas sin control de antaño.
7. Más fallas (por síndrome de “codificar a lo bestia”).

8. Prototipos pueden no escalar, un problema mayúsculo.

9. Funciones reducidas (por “timeboxing”).

Dependencia en componentes de terceros: funcionalidad de más o de menos, problemas legales.

PROGRAMACION EXTREMA XP

HISTORIA

La programación extrema o eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y

más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

INTRODUCCION

Es una 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. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

¿QUÉ ES PROGRAMACIÓN EXTREMA O XP?

- Metodología liviana de desarrollo de software.
- Conjunto de prácticas y reglas empleadas para desarrollar software.
- Basada en diferentes ideas acerca de cómo enfrentar ambientes muy cambiantes.

- Originada en el proyecto C3 para Chrysler.

- En vez de planificar, analizar y diseñar para el futuro distante, hacer todo esto un poco cada vez, a través de todo el proceso de desarrollo.

OBJETIVOS.

- Establecer las mejores prácticas de Ingeniería de Software en los desarrollo de proyectos.

- Mejorar la productividad de los proyectos.

- Garantizar la Calidad del Software desarrollando, haciendo que este supere las expectativas del cliente.

CONTEXTO XP

- Cliente bien definido

- Los requisitos pueden (y van a) cambiar.

- Grupo pequeño y muy integrado (máximo 12 personas.

- Equipo con formación elevada y capacidad de aprender.

CARACTERÍSTICAS XP

- Metodología basada en prueba y error.
- Fundamentada en Valores y Prácticas.
- Expresada en forma de 12 Prácticas–Conjunto completo–Se soportan unas a otras–Son conocidas desde hace tiempo.

VALORES XP

- Simplicidad XP propone el principio de hacer la cosa más simple que pueda funcionar, en relación al proceso y la codificación. Es mejor hacer hoy algo simple, que hacerlo complicado y probablemente nunca usarlo mañana.
- Comunicación Algunos problemas en los proyectos tienen origen en que alguien no dijo algo importante en algún momento. XP hace casi imposible la falta de comunicación.
- Realimentación retroalimentación concreta y frecuente del cliente, del equipo y de los usuarios finales da una mayor oportunidad de dirigir el esfuerzo eficientemente.

- Coraje El coraje (valor) existe en el contexto de los otros 3 valores.(si funciona…mejóralo).

EL ESTILO XP

- Está orientada hacia quien produce y usa el software.
- Reduce el costo del cambio en todas las etapas del ciclo de vida del sistema.
- Combina las que han demostrado ser las mejores prácticas para desarrollar software, y las lleva al extremo.

PRÁCTICAS BÁSICAS DE LA PROGRAMACIÓN EXTREMA

Para que todo esto funcione, la programación extrema se basa en doce "prácticas básicas" que deben seguirse al pie de la letra. Dichas prácticas están definidas (en perfecto inglés) en www.xprogramming.com/xpmag/whatisxp.htm. Aquí tienes un pequeño resumen de ellas.

- Equipo completo: Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.
- Planificación: Se hacen las historias de usuario y se planifica en qué orden se van a hacer y las mini-versiones. La planificación se revisa continuamente.

- Test del cliente: El cliente, con la ayuda de los desarrolladores, propone sus propias pruebas para validar las mini-versiones.

- Versiones pequeñas: Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.

- Diseño simple: Hacer siempre lo mínimo imprescindible de la forma más sencilla posible. Mantener siempre sencillo el código.

- Pareja de programadores: Los programadores trabajan por parejas (dos delante del mismo ordenador) y se intercambian las parejas con frecuencia (un cambio diario).

- Desarrollo guiado por las pruebas automáticas: Se deben realizar programas de prueba automática y deben ejecutarse con mucha frecuencia. Cuantas más pruebas se hagan, mejor.

- Integración continua: Deben tenerse siempre un ejecutable del proyecto que funcione y en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse. Es un error mantener una versión congelada dos meses mientras se hacen mejoras y luego integrarlas todas de golpe. Cuando falle algo, no se sabe qué es lo que falla de todo lo que hemos metido.

- El código es de todos: Cualquiera puede y debe tocar y conocer cualquier parte del código. Para eso se hacen las pruebas automáticas.

- Normas de codificación: Debe haber un estilo común de codificación (no importa cuál), de forma que parezca que ha sido realizado por una única persona.

- Metáforas: Hay que buscar unas frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda uno hacer una idea de qué es lo que hace cada parte del programa. Un ejemplo claro es el "recolector de basura" de java. Ayuda a que todos los programadores (y el cliente) sepan de qué estamos hablando y que no haya mal entendidos.

- Ritmo sostenible: Se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días. Al tener claro semana a semana lo que debe hacerse, hay que trabajar duro en ello para conseguir el objetivo cercano de terminar una historia de usuario o mini-versión.

MANEJO COLECTIVO DEL CÓDIGO

VENTAJAS Y DESVENTAJAS DE EXTREME PROGRAMMING

Ventajas:

- Programación organizada.
- Menor taza de errores.
- Satisfacción del programador.

Desventajas:

- Es recomendable emplearlo solo en proyectos a corto plazo.
- Altas comisiones en caso de fallar.

CONCLUSIONES

- Apostolado de metodologías exitosas.
- Aporte de la experiencia práctica a los modelos teóricos.
- Enfoque de conjunto de prácticas como rompecabezas.
- Tecnología en expansión.
- Importancia de revisitar las metodologías desde la experiencia práctica.

METODOLOGIA EMPLEADA EN EL SOFTWARE

- La metodología RAD fue la elegida por nosotros ya que era la más adecuada y la más precisa para nuestro proyecto.

- Modelado de gestión: ¿Qué información conduce el proceso de gestión? La información que tenemos que solicitar para seguir el proceso principalmente seria los datos básicos del cliente con su mascota. ¿Qué información se genera? Para el caso de solicitar una cita ya podremos tener unos datos cabecera de la mascota y tendremos un historial clínico de esta. ¿Quién la genera? La in formación será solicitada por los veterinarios y el sistema tendrá una base de datos que nos guarde toda la información pedida y actualizada con cada cita. ¿A dónde va la información? Al momento de pedir algún tipo de información esta estará almacenada en una base de datos de la veterinaria. ¿Quién la procesó? Esta información no tendrá algún tipo de procesamiento solo estará dispuesta para ser visualizada, actualizada o eliminada si esto se requiere.

- Modelado de datos: En cuanto a los clientes debemos tener una división de tipo de cliente, formas de pago, expedientes, facturación y visualización de todos los productos ofrecidos por la veterinaria. Y debe tener un sistema de información para los clientes y este ofrece un catálogo, fotografías y un contacto con información de la veterinaria

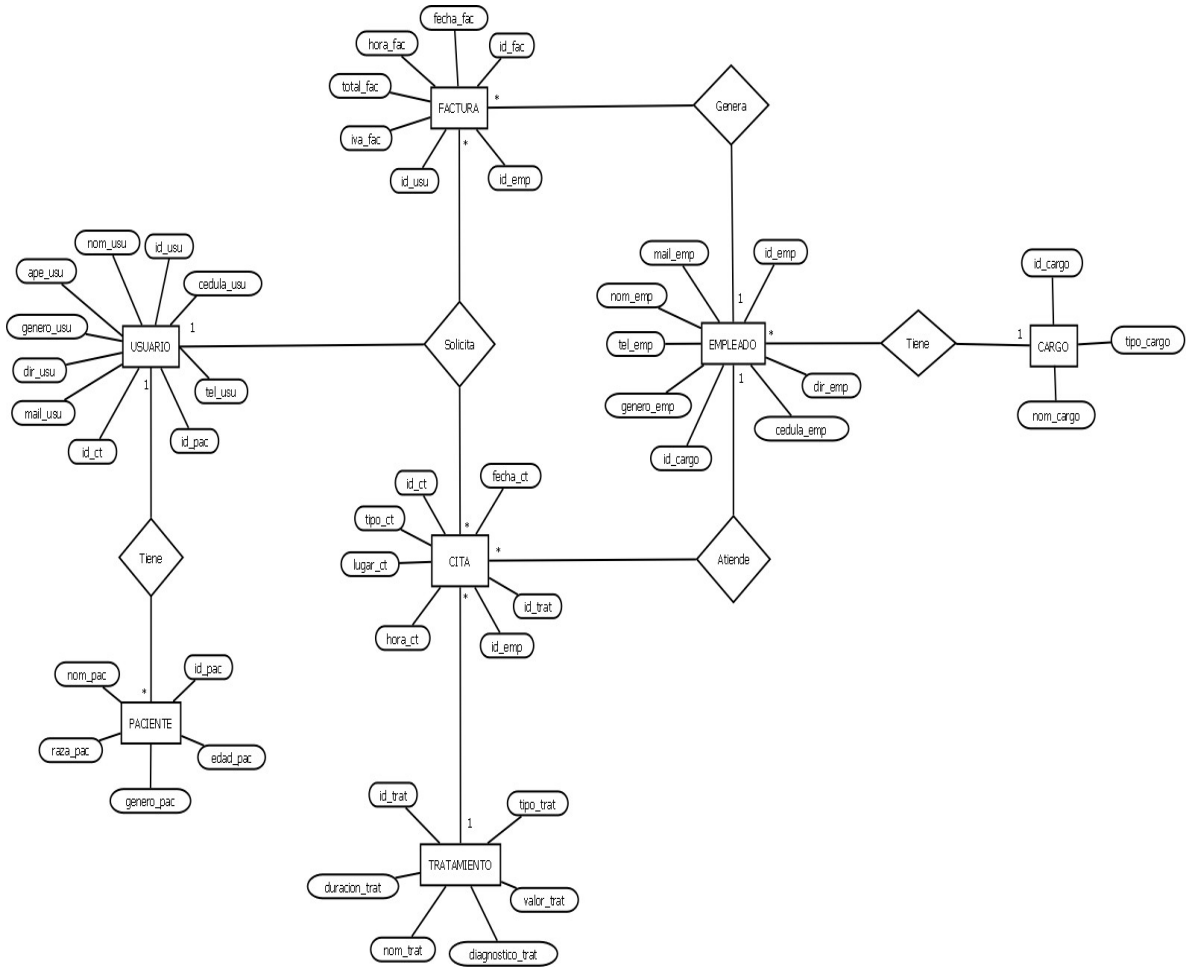
- * Modelado de proceso: Cuando se ingresa a el sistema este le pedirá un usuario y una clave si el cliente no posee una esta le pedirá una serie de información para habilitarle al usuario la posibilidad de manejar sus espacios para solicitar un cita o algún otro tipo de información, pero si solo se desea tener información fotografías o algún tipo de contacto lo podrá hacer sin necesidad de tener un usuario en el sistema.

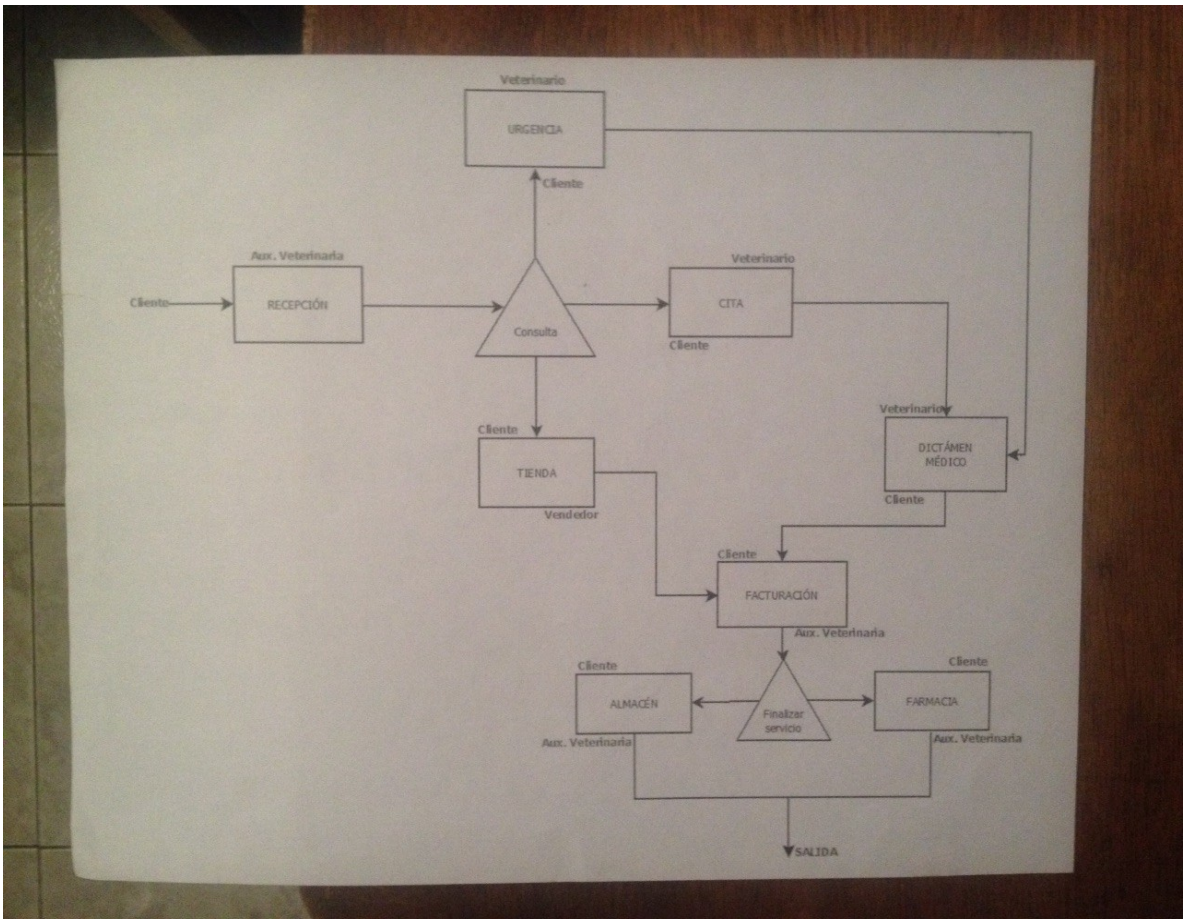
- * Generación de aplicaciones: Se podría generar una aplicación que nos pueda apoyar con la asignación de citas, también para la visualización de estas mismas y por ultimo para la cancelación por si el cliente no puede asistir.

- * Pruebas de entrega: Las pruebas que se pueden ejecutar antes de hacer la entrega, serian por parte del sistema como tal se podría dividir en tres instancias una todo el entorno grafico de la página otra seria la ejecución de la base de datos que esta nos permita guardar modificar y eliminar y la más importante la elaboración del módulo para que nos permita asignar citas.

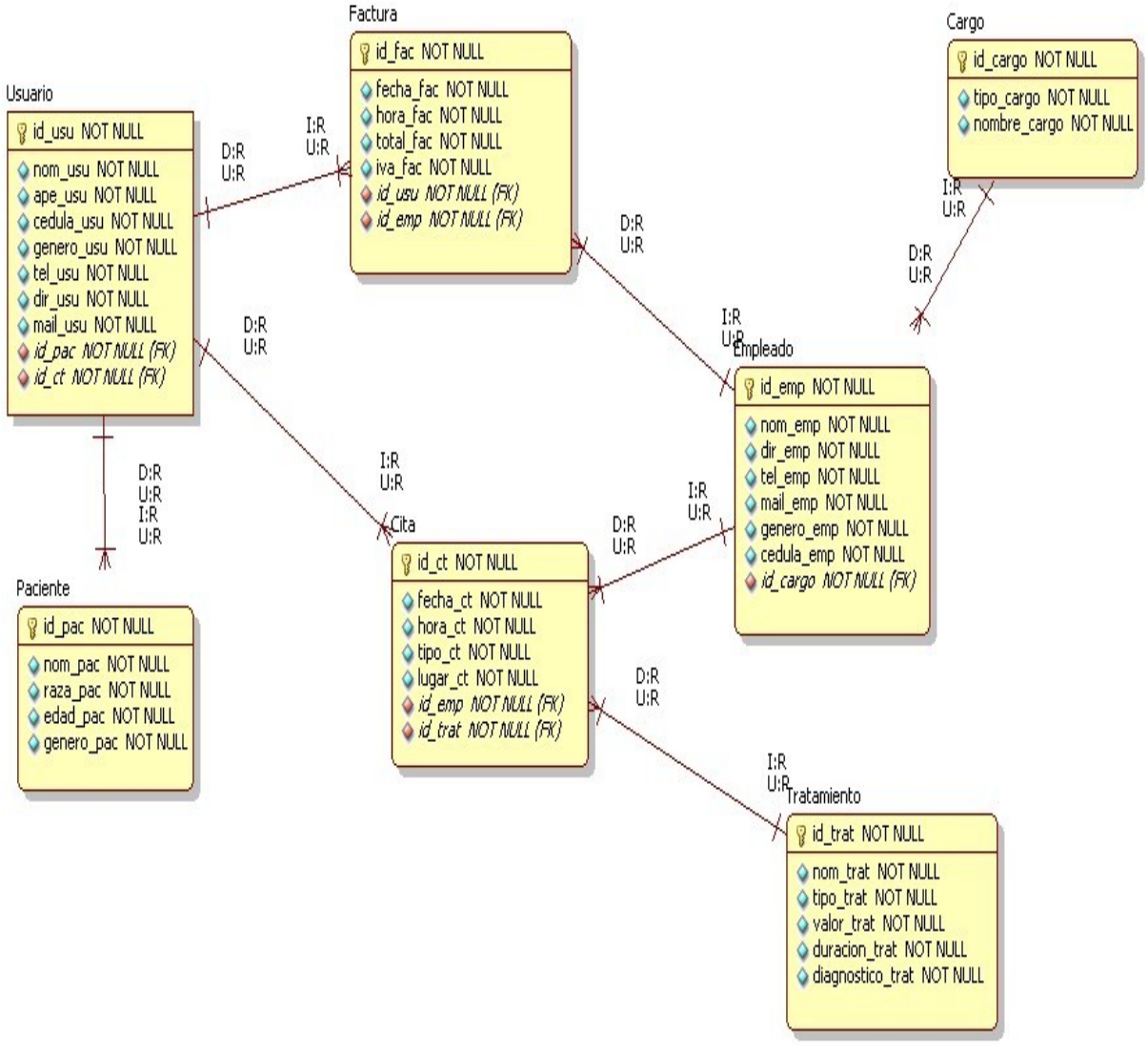
MODELOS

FLUJOGRAMA DE PROCESOS


fecha_fac
hora_fac
id_fac
total_fac
FACTURA
iva_fac
id_usu
id_emp
Genera
nom_usu
id_usu
ape_usu
cedula_usu
genero_usu
USUARIO
dir_usu
tel_usu
mail_usu
id_ct
id_pac
Solicita
mail_emp
id_emp
nom_emp
tel_emp
EMPLEADO
genero_emp
dir_emp
cedula_emp
id_cargo
Tiene
id_cargo
CARGO
tipo_cargo
nom_cargo
id_ct
fecha_ct
tipo_ct
lugar_ct
CITA
id_trat
hora_ct
id_emp
Atiende
Tiene
nom_pac
id_pac
PACIENTE
raza_pac
edad_pac
genero_pac
id_trat
tipo_trat
TRATAMIENTO
duracion_trat
valor_trat
nom_trat
diagnostico_trat


Veterinario
URGENCIA
Cliente
Aux. Veterinaria
Cliente
RECEPCIÓN
Consulta
Veterinario
CITA
Cliente
Veterinario
DICTÁMEN MÉDICO
Cliente
Cliente
TIENDA
Vendedor
Cliente
FACTURACIÓN
Aux. Veterinaria
Cliente
ALMACÉN
Aux. Veterinaria
Finalizar servicio
Cliente
FARMACIA
Aux. Veterinaria
SALIDA

MODELO RELACIONAL



VIABILIDAD O FACTIBILIDAD

Es el proceso de análisis de viabilidad al estudio que intenta conocer previamente el éxito o fracaso de un proyecto. Para lograr esto se necesitan conocer datos empíricos que pueden llegar a ser variables, los cuales se dan a ilustrar a través de diversos tipos de investigaciones (estudios, encuestas, estadísticas, etc.)

El medio factible son las posibilidades que tiene de lograrse el desarrollo de un determinado proyecto. El análisis de factibilidad es el estudio que realiza una empresa para determinar si el negocio que se propone será de gran interés al público o de lo contrario será un fracaso total, y cuáles serán los medios estratégicos que se deben desarrollar para que el proyecto sea un éxito.

TECNICA

El sistema de gestión web SHARP-PEI, requiere un software y un hardware con los elementos básicos de un equipo cómputo, podrá ser implementado en móvil o equipo de escritorio, que cuente con un sistema operativo Microsoft Windows siete en cualquier versión de 64 y 32 bits, Procesador Intel corre i3, y una memoria RAM de 2 gb, es necesario tener un hosting o dominio con acceso a internet.

HUMANA

Para el manejo adecuado del sistema de gestión web SHARP-PEI se realizaran capacitaciones al personal administrativo y trabajadores de la veterinaria, se indicara el manejo, las utilidades y comodidades que permitirá el sistema.

LEGAL

SHARP-PEI es un software desarrollado en herramientas de uso libre; sin embargo para su ejecución, es necesario tener en cuenta los derechos de autor, la compra de su licencia para su utilización legal, al igual que los sistemas operativos de los equipos en los cuales se ejecute.

FINANCIERA

La veterinaria SHARP-PEI va hacer una inversión de cuatro millones aproximadamente que es el valor del software únicamente, ya que cuenta con los equipos y la infraestructura adecuada para su funcionamiento al igual que un plan de internet para el uso del sistema.

COSTOS

Fijos:

Servicios Públicos:

Agua: 40.000 x 3: 120.000

Luz: 50.000 x 3: 150.000

Internet: 50.000 x 3: 150.000

Salario Mínimo: 800.000 x 3: 2.400.000

Total: 2.820.000

Variables:

Transportes: 90.000 x 3: 270.000

Impresiones: 15.000 x 3: 45.000

Fotocopias: 10.000 x3: 30.000

Suministros: 200.000

Total: 545.000

Utilidad:

Total: 3.365.000

20% del total: 673.000

Valor del proyecto

3.365.000 + 673.000: 4.038.000

MATRIZ DOFA

	LISTA DE FORTALEZAS	**LISTA DE DEBILIDADES**
FACTORES INTERNOS FACTORES EXTERNOS	**F1**: BUENA IMAGEN **F2**: INNOVADOR DISEÑO **F3**: FACIL ADAPTACION PARA EL CLIENTE **F4**: CLARIDA DE DATOS **F5:** SE ENCUENTRA UNA NECESIDAD **F6:** INTERES POR LAS VETERINARIAS **F7:** SE IMPLEMENTAN VARIOS CAMPOS	**D1**: POCO ACOGIMIENTO CON EL SECTOR **D2:** EL SISTEMA ESTARA MEDIDO CON LOS CLIENTES **D3:** OCURRENTEMENTE PODRIA FALLAR **D4:** SIN RED NO FUNCIONARIA **D5:** EQUIPOS EFICIENTES
LISTA DE OPORTINIDADES **O1:** MEJORAR LOS INGRESOS DE LA VETERINARIA **O2:** DAR A CONOCER TODOS LOS SERVICIOS OFRECIDOS POR LA VETERINARIA **O3**. ESTAR SIEMPRE A LA PAR CON LO PEDIDO **O4:** TENER UN SOFTWARE SENCILLO PERO UNICO **O5:** TRABAJAR EN RED **O6:** AHORRAR TIEMPO Y ESPACIOS **O7:** MEJORAR EL RENDIMIENTO DE LA EMPRESA	**FO** **ESTRATEGIA PARA MAXIMIXAR TANTO LAS F COMO LAS O** **1: TENER UN SOFTWARE UNICO Y APETECIDO QUE PUEDA SATISFACER LAS NECESIDADES DE LOS CLIENTES Y SU FIDELIDAD.**	**DO** **ESTRATEGIA PARA MINIMIZAR LAS D Y MAXIMIZAR LAS O** **1: PODER SUPLIR TODAS LAS DEBILIDADES POR MEDIO DE REQUERIMIENTOS, ASI DAR LA SOLUCION A TODO LO QUE SE NOS PRESENTE QUE PUEDA INTEFERIR EN LA EVOLUCION DEL SOFTWARE.**
LISTA DE AMENAZAS **A1:** ESCASES DE VISITAS A LA PAGINA **A2:** POCO INTERES DE LOS CLIENTES **A3:** OFERTAS DE OTROS SISTEMAS **A4:** FALLAS DE INTERNET **A5:** VIRUS EN LA RED **A6:** FALLAS DE SEGURIDAD	**FA** **ESTRATEGIA PARA MINIMIZAR LAS A Y MEJORAR LA F** **1: TENER UN SISTEMA SEGURO INMUNE DE ALGUN ATAQUE CIBERNETICO Y DE ALGUA FALLA TECNICA.**	**DA** **ESTRATEGIA PARA MINIMIZAR LAS D COMO LAS A** **1: PODER BRINDAR UN SISTEMA DE FACIL MANEJO PARA EL ADMINISTRADOR COMO PARA LOS CLIENTES QUE TENGAN ACCESO A ESTE.**

GLOSARIO GENERAL

CORREO ELECTRÓNICO:

Servicio de internet que permite el intercambio rápido de mensajes entre personas remotas que no necesariamente han de estar conectadas a la vez. Para poder hacer uso, ambas personas deben disponer de una cuenta, ofrecida por un proveedor de estos servicios

DESCARGAR:

Transferir información desde un ordenador de la red Internet al ordenador propio. También se le suele llamar "bajar" o "download".

DISCO DURO:

Disco que se encuentra en el interior de la CPU y que almacena, tanto la información generada por el usuario, como los archivos necesarios para que los programas funcionen

DOMINIO DE INTERNET:

Es un grupo de entre dos y tres letras que agrupa a un conjunto de servidores con ciertas características comunes. Forma parte del nombre de dominio, que aparece en la dirección de las páginas web

ENLACE:

Elemento de una página web que da acceso a otro documento (o a otra parte del mismo documento) al hacer clic sobre él con el botón izquierdo del ratón. Es la base del acceso a la información en la World Wide Web

HARDWARE:

Cualquiera de los elementos físicos que componen un ordenador: disco duro, placa base, tarjeta gráfica, puertos...

HTML:

Lenguaje de marcas de hipertexto. Es el lenguaje en el que están escritas las páginas web. Realmente se trata de un texto en el que hay insertadas etiquetas que comienzan con el símbolo < y acaban con los símbolos />

INTERNET:

Es una red de ordenadores conectados entre sí que intercambian información a través de las líneas telefónicas. Ofrece multitud de servicios como la world wide web (www), la transferencia de ficheros, las charlas en tiempo real o chats, los foros, el correo electrónico.

MEMORIA RAM:

Parte de la memoria de un ordenador en la que éste almacena información de modo temporal y automático para poder realizar sus operaciones.

NAVEGADOR:

Programa que interpreta el código (HTML y más) en el que están escritas las páginas web y nos las muestra tal y como las vemos en el monitor

ON-LINE:

Estado de un ordenador cando está conectado a internet. La traducción literal es "en línea". Esta expresión se usa cuando se habla de actividades en las que se obtiene una respuesta del servidor que nos da acceso a internet.

PÁGINA WEB:

Es un archivo, escrito en código HTML, que se encuentra en el disco duro de un ordenador conectado a la red internet. Estos archivos se transfieren por la línea telefónica y, si se ven a través de un programa navegador (como Internet Explorer o FireFox) muestran texto, imágenes, animaciones, sonidos, vídeos... Pero lo más característico es que contienen enlaces (también llamados vínculo o hipervínculos) a otras páginas o documentos, de manera que podemos ir "saltando" por la información que nos interesa.

PASSWORD:

Literalmente "palabra de paso". También se le llama contraseña. Es una palabra, conocida sólo por el usuario, que le permite el acceso y uso a zonas privadas, bien de la web, bien de programas específicos-

PROCESADOR:

Circuitos electrónicos incluidos en una pastilla que ejecutan las instrucciones básicas de un ordenador. Es el núcleo que le permite realizar todas las operaciones que nosotros utilizamos. También se le llama microprocesador, debido a su pequeño tamaño.

WINDOWS:

Sistema operativo de Microsoft, que ha ido evolucionando en diferentes versiones para los distintos tipos de ordenadores.

AUTOMATIZAR:
Aplicar la automática a un proceso, a un dispositivo, etc.

ORGANIGRAMA:
Sinopsis o esquema de la organización de una entidad, de una empresa o de una tarea.

WEBGRAFIA

- http://www.qvet.net/que-es-qvet.aspx
- http://www.vetter.com.ar/
- http://metodologiarad.weebly.com/
- http://es.slideshare.net/doogyrm/marco-teorico-16514695
- http://es.slideshare.net/LisPater1/metodologias-agiles-xp?related=1
- http://procesosdesoftware.wikispaces.com/METODOLOGIA+XP
- http://es.wikipedia.org/wiki/Desarrollo_en_espiral
- http://www.compute-rs.com/es/consejos-362625.htm
- http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema03.pdf
- http://es.wikipedia.org/wiki/Software#Proceso_de_creaci.C3.B3n_del_software
- http://148.202.148.5/cursos/cc321/fundamentos/unidad1/espiral.htm
- http://spanishpmo.com/index.php/ciclos-de-vida-modelo-de-cascada/
- http://metodologiasistemasinf.wikispaces.com/Met%C3%B3do+Clasico+del+ciclo+de+vida+para+el+desarrollo+de+sistemas+de+informaci%C3%B3n
- http://ingenieriadesoftwaretdea.weebly.com/ciclo-de-vida-orientado-a-objetos.html
- http://procesosdesoftware.wikispaces.com/METODOLOGIA+RUP
- http://metodologiarad.weebly.com/
- http://ingenieriadesoftware.mex.tl/52753_XP---Extreme-Programing.html