
**Facultad 5**

# BUSCADOR DE ACTIVOS REUTILIZABLES DE SOFTWARE.

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

**Autor:** Marcel Bauta González.

**Tutor:** Ing. Raúl Pérez-Alejo Neyra.

**La Habana, Junio de 2012.**

# Declaración de autoría

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de junio del año 2012.

| | |
|---|---|
| Marcel Bauta González | Ing. Raúl Pérez-Alejo Neyra |
| Firma del Autor | Firma del Tutor |

**Tutor:** Ing. Raúl Pérez-Alejo Neyra.

**Ciudadanía:** Cubana.

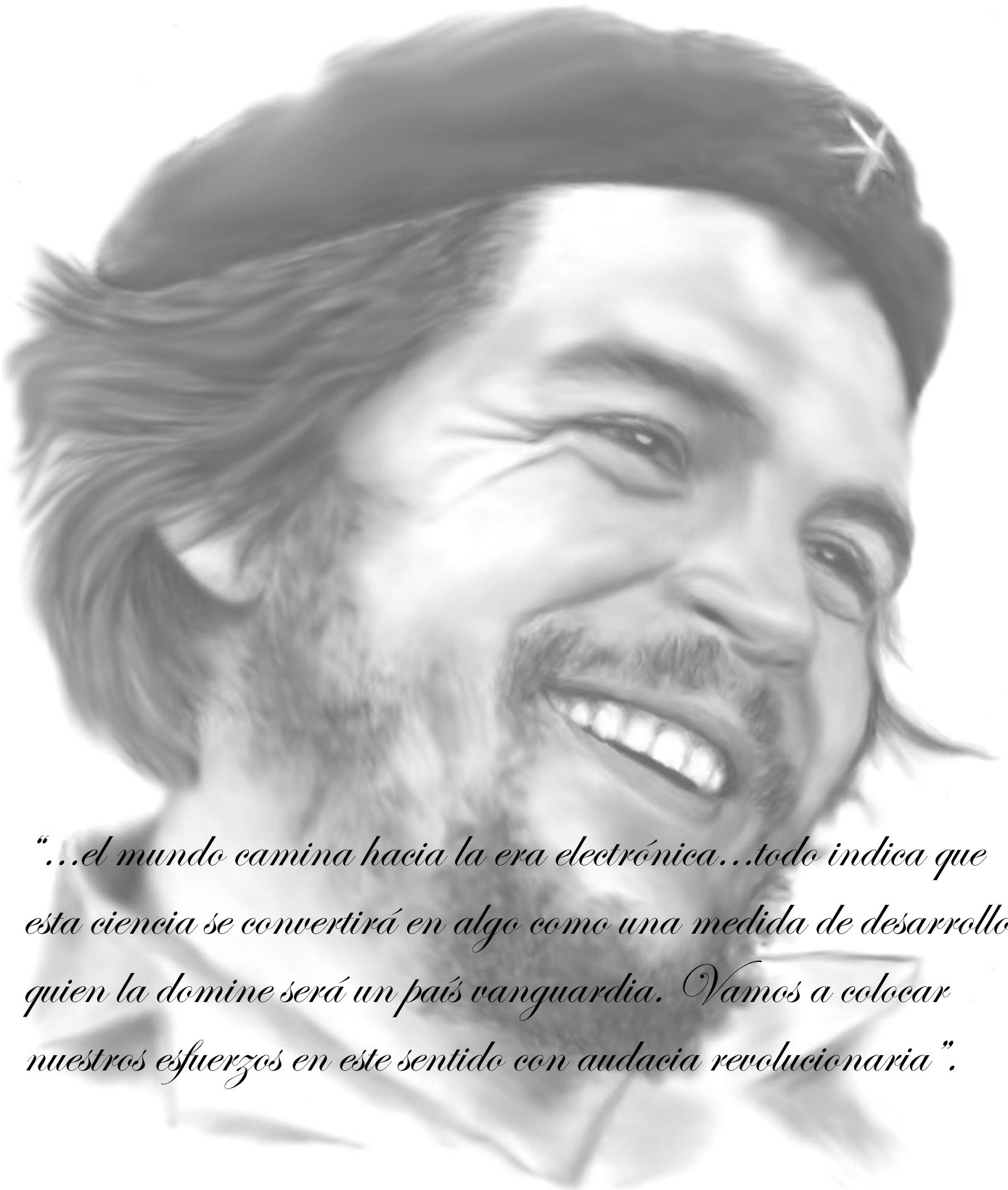
**Institución:** Universidad de Ciencias Informáticas (UCI).

**Título:** Ingeniero en Ciencias Informáticas.

**Categoría Docente:** Instructor.

**E-mail:** rperez-alejo@uci.cu

Graduado de Ingeniero en Ciencias Informáticas en la Universidad de Ciencias Informáticas. Lleva trabajando 4 años en el Centro de Informática Industrial (CEDIN) y tiene categoría docente de Instructor. En ese período ha desempeñado varias tareas frente a Proyectos nacionales y de exportación. Su trabajo ha estado vinculado al mantenimiento y gestión de componentes que se generan en el centro donde trabaja, así como a la Dirección Técnica de la Infraestructura Productiva para temas relacionados con componentes de software.



*"…el mundo camina hacia la era electrónica…todo indica que esta ciencia se convertirá en algo como una medida de desarrollo; quien la domine será un país vanguardia. Vamos a colocar nuestros esfuerzos en este sentido con audacia revolucionaria".*

A la memoria de mi hermano Yusmany.

A mis padres Marcel y Martha por todo su sacrificio y la confianza que siempre han depositado en mí. A ellos que han sabido guiarme, educarme y formarme. Por ser un ejemplo de dedicación, responsabilidad, entrega, y sobre todo por ser mi mayor orgullo.

A mi hermano menor Rafael que ya casi es un hombre, por ser mi amigo incondicional, por el cariño que me brinda y lo que significa en mi vida.

A mis abuelos Eyde y Crescente por ser mis ángeles de la guarda y mis paradigmas de personas.

Dedico especialmente este trabajo de diploma a mi novia Marlen por regalarme todo su amor, cariño y comprensión. Por haber sido más que mi pareja, mi mejor amiga y lo que hemos compartido juntos, por su ayuda en todo momento, y por haber estado a mi lado durante estos cinco años sin importar las circunstancias.

Un agradecimiento especial a la Revolución por darme la oportunidad de estudiar una carrera universitaria en esta magnífica escuela. A mi tutor Raúl por aclararme las dudas, por su apoyo incondicional durante el desarrollo del presente trabajo, por su amistad, su optimismo y paciencia.

A todos los amigos del preuniversitario y la universidad que son muchos y no podría mencionarlos a todos, en especial a Vladimir, Daniel, Nivardo, José Luis, Carlos, Yasmany, Osmay, Rubén, Pedro, Julio, Pablo, Enrique, Alain, Dausbel, Leyannis, Claudia, Yanai, Olivia, Ana Celia, Yusemi y Edumis por haberse convertido en mi familia durante estos cinco años.

A todos los profesores que han contribuido en mi formación, transmitiéndome sus conocimientos, en especial a Yoan y Gerandys.

Para mi familia numerosa y querida, mis primos, tíos, abuelos, quienes fueron mis primeros maestros y los más grandes críticos a la hora de cualquier flaqueza o desmotivación.

Para Chela, Janyer y Surema todo mi cariño y agradecimiento pues siempre han cuidado de mi papá y de mí.

A mis amigos de Holguín, Alberto y Jorgito, los que me demostraron que los amigos verdaderos son como las estrellas, a veces no se ven pero siempre sabes que estarán ahí.

A mis amigos del fútbol, ojalá y la vida nos reúna de nuevo en otras canchas, ustedes siempre serán los mejores de la universidad.

A todos gracias por aportar su granito de arena y ayudar a convertirme en un hombre de bien.

En la Universidad de la Ciencias Informáticas (UCI) por la gran cantidad de proyectos existentes se requiere de altos niveles de producción para poder satisfacer las demandas de los clientes cumpliendo con el tiempo de entrega y la calidad de los productos. Debido a estos grandes volúmenes de producción se estipula la implantación del modelo Líneas de Productos de Software (LPS) cuyo objetivo principal es reducir el tiempo, esfuerzo, costo y complejidad de crear y mantener los productos. Este modelo incluye, fomenta y requiere del uso del repositorio de activos de software reutilizables, que en su actual implementación, las búsquedas multiplataforma se realizan de forma manual navegando por las páginas web donde se visualizan los ficheros. Este es un proceso lento y se tiene el riesgo de no abarcar el repositorio en su totalidad. El presente trabajo tiene como objetivo la implementación de un software mediador para la búsqueda de activos de software, dentro de un repositorio.

**Palabras clave:**

LPS, activos, repositorio, mediador.

**Abstract:**

At the University of Informatics Sciences (UCI) for the large number of existing projects require high levels of production in order to meet customer demands compliance with delivery time and product quality. Because of this high-volume production is provides the implementation of the model of Software Product Lines (SPL) whose main objective is to reduce the time, effort, cost and complexity to creating and maintaining the products. This model includes, promotes and requires the use of a repository of reusable software assets, which in its current implementation, multi-platform searches are performed manually browsing the web pages where files are displayed. This is a slow process and there is a risk of not covering the entire repository. This work has as objective to implement a mediator software to search for software assets inside a repository.

**Keywords:**

SPL, assets, repository, mediator.

## ÍNDICE DE CONTENIDOS.

## ÍNDICE DE FIGURAS.

## INTRODUCCIÓN.

La informática constituye uno de los más grandes logros de la humanidad. Gracias a los incontables avances y al auge del uso de las Tecnologías de la Informática y las Comunicaciones (TIC[1]) en la actualidad existen empresas dedicadas a la producción de software, convirtiéndose en uno de los renglones fundamentales de la economía en varios países.

Por la creciente demanda de los clientes en el mercado del software se requieren altos niveles de producción para poder satisfacerla cumpliendo con los tiempos de entrega y la calidad de los productos. La ingeniería de software basada en la reutilización, es una disciplina que nació de la necesidad del desarrollo de nuevos productos a partir de otros existentes.

La reutilización de componentes de software representa una forma económica de producir con mayor calidad y rapidez. Entre las ventajas se destacan la reducción del tiempo en el mercado y del desarrollo de los productos, el incremento de su calidad, aumento de la satisfacción de los clientes y lograr una alta escala en la productividad [1].

Basadas en el principio de la reusabilidad surgen las Líneas de Productos de Software (LPS) que constituyen un modelo de desarrollo de software basado en la creación de varios productos similares a partir del «ensamblaje» de componentes pre-fabricados [2]. Otras bibliografías la denominan “portafolio de productos estrechamente relacionados con variaciones en sus características y funciones” [3].

Debido al uso de las LPS, se logra un incremento del número de compañías de alta tecnología en el mercado, fomentando la reutilización de todos los artefactos en el ciclo de vida de desarrollo, y así acortar el tiempo de desarrollo y mantenerse competitivo en el mercado [4]. El objetivo principal de las LPS es “reducir el tiempo, esfuerzo, costo y complejidad de crear y mantener los productos de la línea” [5].

En la siguiente figura se muestra gráficamente como el modelo minimiza los gastos de producción para una cantidad igual de productos que los enfoques de desarrollo tradicionales, una vez que se ha llegado a la madurez del proyecto y los volúmenes de producción son mayores.

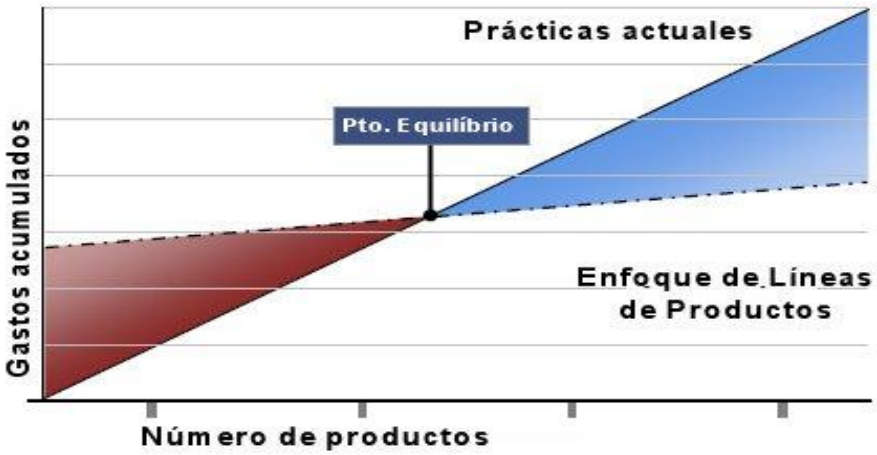



***Ilustración 1LPS vs prácticas actuales.***

Nuestro país ha creado organismos y empresas para solventar la necesidad creciente que existe de desarrollar productos informáticos, ya sea para la comercialización y cubrir la demanda nacional e internacional. Entre las principales entidades productoras de software se encuentra la Universidad de Ciencias Informáticas, (UCI) en la cual, existen diferentes centros especializados; uno de ellos es el Centro de Informática Industrial (CEDIN[2]) que funciona como un centro generador de soluciones de software, desarrollando tecnologías, productos y servicios asociados a la Informática Industrial. Como parte de la estructura de este centro se encuentra el Grupo de Integración encargado de integrar las soluciones en los dominios del negocio, llevar la gestión del repositorio de componentes así como implantar las soluciones de los productos informáticos producidos por el centro.

La UCI estimula la implantación de las LPS como modelo rector para la producción en sus centros especializados y toma como referencia las innumerables ventajas que aportan. Dicho modelo incluye técnicas para la mejora continua de la calidad del producto y principalmente fomenta y requiere del uso del repositorio de activos de software reutilizables.

La actual implementación del repositorio de activos reutilizables solo permite buscar softwares en la plataforma GNU/Linux mediante herramientas de gestión de paquetes como Synaptic[3], las búsquedas se realizan de manera restringida a categorías muy generales y no se pueden hacer búsquedas anidadas. Para las plataformas Windows y Linux existe un método común de búsqueda mediante la web, que es

más limitado pues se realiza dentro de los repositorios de forma manual navegando por las páginas que visualizan los ficheros donde físicamente se guardan los activos de manera alfabética. Este proceso es lento y el resultado depende de la paciencia del interesado, se corre el riesgo de no abarcar todo el repositorio por su potencial extensión y los resultados no serán los que mejor se adapten a las necesidades debido a la posible conformidad con el resultado de una búsqueda.

A partir de la **situación problémica** descrita se plantea el siguiente **problema a resolver**: ¿Cómo optimizar el proceso de búsquedas en un repositorio de activos reutilizables de software?

Se define como **objeto de estudio de la investigación** el repositorio de activos de software, y como **campo de acción** la gestión de búsqueda dentro del repositorio de activos de software.

En correspondencia con el problema planteado, se determina como **objetivo general** desarrollar un software mediador para la búsqueda de activos de software.

Para dar cumplimiento al objetivo general son trazados los siguientes **objetivos específicos:**

- Fundamentar la investigación, mediante la elaboración del marco teórico.
- Estudiar las tecnologías definidas para el cumplimiento del objetivo.
- Realizar el análisis y diseño de la solución.
- Implementar la propuesta de solución.
- Aplicar pruebas al producto para validar su correcto funcionamiento.

Para el diseño de la investigación se plantea la siguiente **idea a defender**: con el desarrollo del software mediador se optimizará el proceso de búsquedas, dentro de un repositorio de activos reutilizables de software.

Para dar cumplimiento a los objetivos propuestos se utilizan los siguientes **métodos de investigación.**

**Métodos Teóricos.**

- **Histórico–Lógico:** Para realizar el estudio de las tendencias históricas y actuales en el desarrollo de herramientas de búsqueda dentro de repositorios de paquetes.

- **Análisis-Síntesis:** Para hacer el estudio de las teorías y documentos relacionados con las herramientas de búsqueda dentro de repositorios y extraer los elementos y conceptos más significativos sobre la investigación.

**Métodos Empíricos.**

- **Experimento:** Para establecer las diferentes pruebas con el propósito de determinar la fiabilidad del sistema y el mejoramiento de la usabilidad del mismo, a través de la detección de errores.

Como **posible resultado** se contará con un software mediador donde el resultado de una búsqueda serán todos los componentes que respondan a una o varias clasificaciones. También permitirá la clasificación de activos reutilizables de software y realizará una visualización de las etiquetas mediante la nube.

**Estructura capitular.**

El contenido a desarrollar en el presente trabajo está estructurado en 3 capítulos:

**Capítulo 1: Fundamentación teórica.** Se precisan un conjunto de conceptos y definiciones que le ofrecen al lector una panorámica del contexto de la investigación, incluye un estudio sobre el estado del arte del tema que se aborda para conformar un sólido basamento teórico. Se realiza un análisis sobre la gestión de búsquedas dentro de un repositorio de activos, su definición e importancia. Por último se hace referencia a las tecnologías y metodologías seleccionadas, que serán de utilidad para su desarrollo práctico.

**Capítulo 2: Exploración, planificación y diseño.** Durante la Fase de Exploración se describen las Historias de Usuarios (HU) en correspondencia con cada uno de los requisitos funcionales identificados. Se hace una descripción de todos los artefactos generados mediante las HU. Durante la Fase de Planificación se realiza la estimación del esfuerzo por cada HU, el plan de iteraciones y de duración de cada una y de entregas con las propuestas de liberación de las versiones de la aplicación.

**Capítulo 3: Implementación y pruebas.** Durante la Fase de Implementación se describen las tareas de programación en correspondencia con cada una de las Historias de Usuarios identificadas. Se hace una descripción de los estándares de codificación empleados en el desarrollo del sistema. Durante la Fase de Pruebas se realizan las pruebas unitarias, funcionales y de aceptación.

# CAPÍTULO 1
# FUNDAMENTACIÓN TEÓRICA.

**Introducción.**

En el mundo actual, el constante desarrollo y evolución de las tecnologías informáticas crece de forma exponencial. Nuestro país fiel a su política de soberanía nacional apuesta por lograr una independencia tecnológica utilizando software y aplicaciones libres, por los beneficios que significarían para nuestra economía nacional, evitándose así el gasto innecesario de millones de dólares en conceptos de compra de software privativo y licencias. Teniendo en cuenta estos principios, desde hace cuatro años a nivel de Universidad se definió el empleo de la herramienta web de gestión de proyectos Redmine[4] que integra Rails como framework[5] de desarrollo, utiliza Ruby como lenguaje de programación y PostgreSQL como gestor de base de datos. Este trabajo de diploma forma parte de una pirámide investigativa cuyo objetivo final está orientado a la integración con el Redmine, por lo que para el desarrollo del software mediador se utilizan las mismas tecnologías en la que está implementado el software de gestión.

En el desarrollo del presente capítulo se exponen características de las tecnologías propuestas, se describen las tendencias, herramientas y metodologías usadas para el desarrollo. Se realiza un estudio sobre el estado del arte de la gestión de búsquedas dentro de un repositorio de activos reutilizables y se analizan un conjunto de conceptos y definiciones que le proporcionan al lector una panorámica del contexto de la investigación.

## 1.1 Búsquedas en repositorios existentes.

Es necesario hacer un análisis sobre cómo se realizan las búsquedas, en los principales sistemas de gestión de repositorios, que favorezca el desarrollo del presente trabajo. Se requiere del estudio de estos sistemas, para de ser posible, utilizar funcionalidades similares y tener conocimiento sobre que posibles herramientas y tecnologías utilizar en la aplicación. A continuación se exponen varios de estos ejemplos.

### 1.1.1 SourceForge.

Es un sitio web dirigido al desarrollo colaborativo de Software Libre que actualmente permite, como principal objetivo, gestionar el código fuente de más de 324.000 proyectos [6]. Todo el software alojado en SourceForge (SF) tiene una característica común: es de código abierto, lo cual permite que cualquiera con los conocimientos necesarios pueda acceder al código fuente, colaborar en su desarrollo y aportar nuevas ideas.

La gestión de búsquedas de SF se realiza desde la interfaz principal, permitiendo filtrar proyectos por categorías generales que no pueden ser asociadas, son muy generales. Otra opción de búsqueda es posible mediante la entrada de textos en la parte superior del sitio, este método solo busca en los nombres de los softwares o en la descripción, los resultados se muestran ordenados por algunos criterios como son la relevancia y la popularidad, lo que para el objetivo del sitio es de gran utilidad, pero no para el cumplimiento de los objetivos de esta investigación.

### 1.1.2 Git y Subversion.

Estas herramientas fueron creadas para el trabajo colaborativo en el desarrollo de software, la principal diferencia entre ellas radica en la orientación a ambientes que tienen. La primera fue creada por Linus Torvalds y se ajusta más a desarrollo de manera distribuida geográficamente. La segunda está orientada a un desarrollo centralizado donde las personas participantes están conectadas a un servidor remoto o local, lo que condiciona tener conectividad. En la actualidad estos son las dos herramientas fundamentales que utilizan los proyectos para gestionar los códigos.

Ambas herramientas tienen opciones de búsqueda por revisiones, fechas de cambios o comentarios realizados al salvar el código, estas tres maneras de búsqueda presentan la dificultad de tener que recordar algún dato para iniciar y luego mediante decantación ir acercando el resultado de la revisión esperado. Existen algunas aplicaciones que ayudan de manera visual en este tipo de búsquedas en los repositorios.

Kenai es un gestor de repositorios creado con JRuby[6] y sistema de bases de datos MySQL con soporte para Mercurial, Subversion y Git, que ofrece un sistema para búsquedas de desarrolladores y proyectos mediante nube de etiquetas, permite el etiquetado de proyectos, noticias y errores [7]. Es un sistema de control de versiones libre y de código fuente abierto diseñado específicamente para reemplazar al popular

CVS. Subversion puede acceder al repositorio a través de redes, lo que le permite ser utilizado por personas que se encuentran en distintos ordenadores; es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros [8].

ViewCVS es un script CGI basado en Python para hacer búsquedas en repositorios de CVS y Subversion y desde hace bastante tiempo ha sido la mejor opción para proporcionar una interfaz basada en Web para estos sistemas de control de versiones. ViewCVS permite a los usuarios navegar fácilmente a través de un repositorio Subversion [9]. Para realizar búsquedas, el nivel superior del repositorio es mostrado, con enlaces a cada directorio o archivo a ese nivel. A continuación, se pueden ver los archivos haciendo clic sobre ellos, o descendiendo por los subdirectorios. El aspecto más interesante de esta herramienta es que permite ver una revisión deseada escribiendo su número en un cuadro de texto.

Los tres sistemas relacionados con la búsqueda en los repositorios anteriormente expuestos presentan funcionalidades y características que los hacen muy robustos, confiables y utilizables por los usuarios. Su implantación resuelve las necesidades para los tipos de repositorios específicos que fueron creados, por sí solos no brindan solución a la problemática planteada en la presente investigación, pero entre todos presentan características, que sirven para determinar cuáles son las funciones principales que debe poseer un sistema de búsquedas para que la solución que se proponga sea la más adecuada al contexto del repositorio de activos de software, de acuerdo con los beneficios obtenidos por otras experiencias.

Como resultado del estudio realizado se aprecia que sobresalen tres funciones principales, por las facilidades y características que brindan, a tener en cuenta para el software mediador; que permita la búsqueda, tanto escribiendo palabras que se correspondan a etiquetas, como utilizando la nube y que realice la catalogación de activos mediante categorías y etiquetas.

## 1.2 Componentes y activos de software.

Un componente de software reutilizable (CSR) se define en [10] y [11] como “una pieza [de software] funcional que es liberada independientemente [de otras] y que proporciona acceso a sus servicios a través de sus interfaces”. Szypersky define en [12] un CRS como: “Unidad de composición de aplicaciones de software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio.”

Entre las características esenciales de un CSR definidas en [13] y [14] están:

- **Identificable:** Debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación. Debe ser clara y consistente de modo que facilite su catalogación y búsqueda en repositorios de componentes.
- **Auto contenido:** Un componente no debe requerir de la utilización de otros para finiquitar la función para la cual fue diseñado. Es conveniente que un componente dependa lo menos posible de otros componentes para cumplir su función de forma tal que pueda ser desarrollado, probado, optimizado, utilizado, entendido y modificado individualmente.
- **Reemplazable por otro componente:** Se puede cambiar por nuevas versiones u otro componente que lo mejore.
- **Accesible solamente a través de su interfaz:** El componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.
- **Inmutabilidad de sus servicios:** Las funcionalidades ofrecidas en su interfaz no deben variar, pero su implementación, sí. Estas variaciones no deben afectar la interfaz.
- **Bien documentado:** Un componente debe estar correctamente documentado, incluyendo sus servicios, para facilitar su búsqueda si se quiere actualizar, integrar con otros, adaptarlo, etc.
- **Genérico:** Sus servicios debe servir para varias aplicaciones.
- **Independiente de la plataforma:** Existen diversas plataformas de cómputo de uso frecuente (ej. Windows/Intel, Solaris/Sparc, OSX/PPC, Linux/Intel) y es deseable que un componente pueda ejecutarse en todas ellas, pues existe una amplia gama de lenguajes de programación y herramientas de desarrollo, es natural que encontremos componentes escritos empleando lenguajes y herramientas de la preferencia del programador, por lo tanto es deseable que dichas preferencias no limiten el uso de los componentes.

No obstante, el componente no es la única pieza que se puede volver a utilizar de un software. Existe el término activo de software, el cual abarca todos los artefactos que se generan de un software y que puedan ser reutilizados. Montilva lo clasifica en [15] como “un producto de software diseñado

expresamente para ser utilizado múltiples veces en el desarrollo de diferentes sistemas o aplicaciones", por lo que también pertenecen a esta definición:

- Un componente de software
- Una especificación de requisitos
- Un modelo de negocios
- Una especificación de diseño
- Un algoritmo
- Un patrón de diseño
- Una arquitectura de dominio
- Un esquema de base de datos
- Una especificación de prueba
- La documentación de un sistema

O cualquier otro artefacto que por su naturaleza sea del interés de la organización donde se produjo.

### 1.3 Reutilización de componentes y activos de software.

La reutilización es un proceso inspirado en la manera en que se producen y ensamblan componentes en la ingeniería de sistemas informáticos. La aplicación de este concepto al desarrollo de software no es nueva, las librerías de subrutinas especializadas, comúnmente utilizadas desde la década de los setenta, representan uno de los primeros intentos por reutilizar software [15].

Existen variadas definiciones del término reutilización de software. Somerville afirma en [16] que la reutilización es un enfoque de desarrollo [de software] que trata de maximizar el uso recurrente de componentes de software existentes. Según Sametinger, la reutilización de software es el proceso de crear sistemas de software a partir de software existente, en lugar de desarrollarlo desde el comienzo [17]. Por su parte J. Sodhi y P. Sodhi definen la reutilización de software como el proceso de implementar o actualizar sistemas de software usando activos de software existentes [18]. El análisis de estas definiciones permite establecer una definición más detallada en los siguientes términos:

La reutilización de software es un proceso de la Ingeniería de Software que conlleva al uso recurrente de componentes y activos de software en la especificación, análisis, diseño, implementación y pruebas de una aplicación o sistema de software [15].

### 1.4 Repositorios de activos de software.

Un repositorio, depósito o archivo es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos. El origen de la palabra española repositorio deriva del latín *repositorĭum*, que significa armario o alacena. Este término está recogido en el Diccionario de la Real Academia donde se define como el «lugar donde se guarda algo».

Los repositorios pueden distribuirse sirviéndose de una red informática como Internet o en un medio físico como un disco duro o compacto. Pueden ser de acceso público, o pueden estar protegidos y necesitar de una autentificación previa. A diferencia de los ordenadores personales o de escritorio, los repositorios suelen contar con sistemas de respaldo (Backup) y mantenimiento preventivo y correctivo, lo que hace que la información se pueda recuperar en el caso que la máquina quede inutilizable.

El término “biblioteca de activos de software” o “repositorio de activos de software” se refiere al sistema de información para guardar y gestionar tanto los activos de reutilización como la información que se almacenen de ellos, tales como [19]:

- La descripción general de cada activo.
- El metadato con algunos detalles técnicos.
- Y el empaquetado que se almacena junto al activo, el cual agrupa un conjunto de documentos que se detalla más adelante.

### 1.5 Búsqueda en repositorios de activos de software.

De los procesos definidos en [20] para la explotación de repositorios de activos de software que se muestran en la siguiente ilustración, este trabajo de diploma se centra en el subproceso de búsqueda perteneciente al proceso consumir de la red.

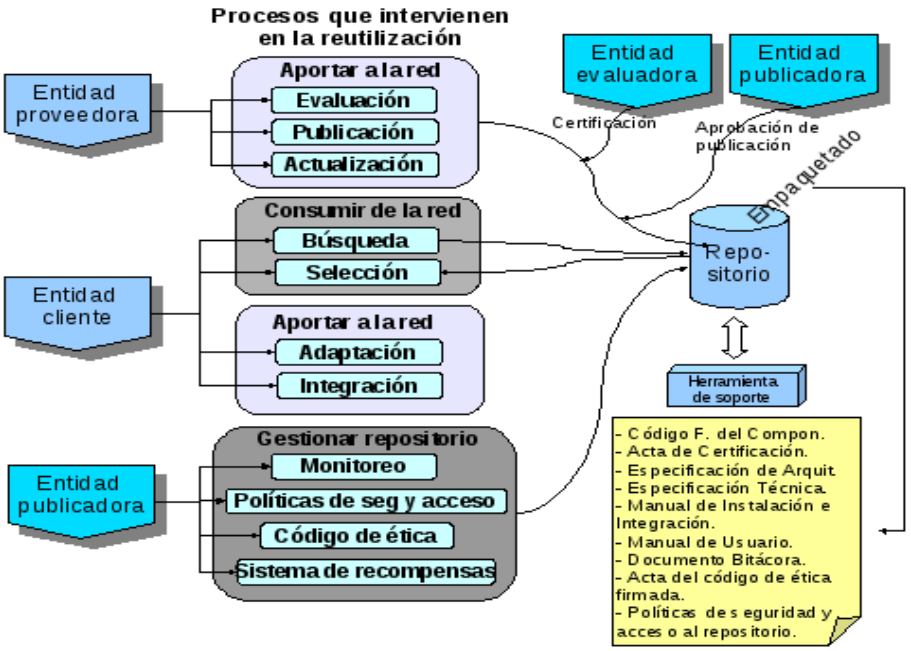



***Ilustración 2 Procesos definidos para la explotación de repositorios.***

La búsqueda es una de las técnicas principales en el desarrollo de software centrados en la reutilización para la localización de activos en el repositorio [21]. Es realizada por la entidad cliente y consiste en revisar entre los repositorios disponibles para la reutilización, cuál o cuáles se ajustan más a las especificaciones del entorno donde será integrado, es decir, cuál satisface los requisitos impuestos tanto por el cliente como por la arquitectura de la aplicación. Para ello, el cliente se auxilia de los metadatos registrados por cada activo, donde podrá encontrar un resumen o descripción de las funcionalidades o servicios que ofrece un activo, o más específicamente, un componente y la parte más significativa de la información técnica del mismo. Generalmente se hace a través de un mediador (programa de búsqueda de componentes en un repositorio) o algún otro algoritmo de búsqueda que implemente el proveedor de activos para ofrecer como servicio agregado, el de “búsqueda de activos”.

Los algoritmos de búsqueda en repositorios de activos de software son una extensión de los tradicionales algoritmos de búsqueda, entre los cuales se encuentran:

(a) basados en palabras clave (keywords) [22].

(b) basados en categorías (facets) [23]. Los paquetes se clasifican por categorías que generalmente son representadas por pequeñas palabras llamadas etiquetas (tags).

(c) basados en conocimiento [24] y [25]. Utilizan información estadística y basada en conocimiento para la selección de componentes.

(d) basados en especificaciones.

La clasificación por categorías propuesta por Prieto-Díaz y Freeman en [23], se basa en búsquedas por aspectos que son extraídas por expertos, para describir las características acerca de los componentes, estas sirven como descriptores, tales como la funcionalidad, la forma de ejecutarse, y los detalles de implementación, entre otros. Esta ha demostrado ser muy eficaz cuando se realiza en repositorios de activos de software, no limita la clasificación a un formato jerárquico general, proporcionando mayor flexibilidad. Las categorías disponibles, sus etiquetas correspondientes y su asociación con los activos se pueden modificar o eliminar. Por estas razones se utiliza este tipo de búsqueda como método en el desarrollo de la aplicación.

### 1.6 Ruby y Rails.

Ruby es un lenguaje de programación creado por el programador japonés Yukihiro "Matz" Matsumoto. Es similar en funcionalidades con otros lenguajes de programación como Lisp, Lua, Dylan, CLU, Python y Perl pero más orientado a objetos. Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre. Como lo indica su propio autor, Ruby es un lenguaje “aparentemente sencillo pero internamente complejo”.

Ruby fue diseñado para un desarrollo rápido y sencillo y cada día este lenguaje va ganando más adeptos. El intérprete y las bibliotecas están licenciados de forma dual (inseparable) bajo las licencias libres y de código abierto GPL[7] y Licencia pública Ruby.

El framework Rails también conocido como Ruby on Rails (RoR) fue creado por David Heinemeier Hansson. Rails es de código abierto y está escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC), trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros framework y con un mínimo de configuración. Rails es un framework para el desarrollo de aplicaciones web, software libre por naturaleza.

Fue liberado por primera vez al público en julio del 2004, y lo implementó en una aplicación orientada a la administración de proyectos llamada Basecamp[8].

### 1.7 PostgreSQL.

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL está derivado del paquete Postgres escrito en Berkeley. Con cerca de una década de desarrollo tras él, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día: ofrece control de concurrencia multi-versión, soporta casi toda la sintaxis SQL (incluyendo subconsultas, transacciones y funciones definidas por el usuario), cuenta también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, Tcl, Python y Ruby). Es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD[9].

### 1.8 Metodologías de desarrollo de software.

Debido al gran auge y desarrollo que ha alcanzado la producción de software a nivel mundial se ha creado una inmensa cantidad de documentación sobre las políticas, procesos y procedimientos para lograr la calidad del producto final. Entre los principales objetivos de las metodologías de desarrollo de software se encuentran cumplir con los requisitos iniciales del problema y minimizar las pérdidas de tiempo en el proceso de desarrollo así como minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de los productos.

#### 1.8.1 RUP.

RUP es un proceso formal: Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). Fue desarrollado por Rational Software[10], y está integrado con toda la suite Rational de herramientas. Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte [26].

RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. RUP describe cómo aplicar efectivamente el desarrollo de software. La utilización de esta metodología de desarrollo se conoce actualmente como "mejor práctica" (del inglés best practice) ya que son utilizados en la industria por organizaciones triunfantes y con resultados exitosos de su uso.

El proceso de software propuesto por RUP tiene tres características esenciales: está dirigido por los Casos de Uso, está centrado en la arquitectura, y es iterativo e incremental. RUP identifica 6 “best practices” con las que define una forma efectiva de trabajar para los equipos de desarrollo de software.

- Gestión de requisitos
- Desarrollo de software iterativo
- Desarrollo basado en componentes
- Modelado visual (usando UML)
- Verificación continua de la calidad
- Gestión de los cambios

### 1.8.2 SCRUM.

Scrum es una metodología ágil de desarrollo de proyectos que toma su nombre y principios de los estudios realizados sobre nuevas prácticas de producción por Hirotaka Takeuchi e Ikujijo Nonaka a mediados de los 80. Aunque surgió como modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de software [27].

Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto. Estas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración [28].

Scrum es una metodología de desarrollo muy simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto y constituye una plantilla para dar forma a los principios ágiles. Es una ayuda para organizar a las personas y el flujo de trabajo; como lo pueden ser otras propuestas de formas de trabajo ágil: Cristal, DSDM, etc.

### 1.8.3 Programación Extrema (XP).

Constituye 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 la 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. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP en [29] sin cubrir los detalles técnicos y de implantación de las prácticas. Entre las principales características de esta metodología se encuentran.

- Permite introducir nuevos requisitos o cambiar los anteriores ágilmente.
- Adecuado para proyectos pequeños y medianos.
- Adecuado para proyectos con alto riesgo.
- Su ciclo de vida es iterativo e incremental.
- Cada iteración dura entre una y tres semanas.
- No produce demasiada documentación acerca del diseño o la planificación.


***Ilustración 3 Programación extrema (XP).***

Teniendo en cuenta las peculiaridades y características de las metodologías analizadas anteriormente y basándose en las particularidades del desarrollo del software a implementar se decide utilizar Programación Extrema como metodología de desarrollo de software, pues posee características y ventajas sobre las demás metodologías ágiles analizadas como son:

- Se consiguen productos usables con mayor rapidez
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo.
- Se atienden las necesidades del usuario con mayor exactitud. Esto se consigue gracias a las continuas versiones que se ofrecen al usuario.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de los test de forma previa a la codificación.
- Gracias al “refactoring” es más fácil el modificar los requerimientos del usuario.
- Se consigue tener un equipo de desarrollo más contento y motivado. Las razones son, por un lado el que la XP no permite excesos de trabajo (se debe trabajar 40 horas a la semana), y por otro la comunicación entre los miembros del equipo que consigue una mayor integración entre ellos.
- Que el software funcione es más importante que la documentación exhaustiva. Desarrollar un software que funciona más que conseguir una buena documentación es fundamental. La regla a seguir es: no producir documentos a menos que sean necesarios de forma inmediata. Estos documentos deben ser cortos y centrarse en lo fundamental.

### 1.9 Lenguaje de modelado UML.

El UML (Lenguaje Unificado para la Construcción de Modelos) se define como un "lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software...". Es un sistema notacional (que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos [30].

El UML es un estándar incipiente de la industria para construir modelos orientados a objetos. Nació en 1994 por iniciativa de Grady Booch y Jim Rumbaugh para combinar sus dos famosos métodos: el de Booch y el OMT (Object Modeling Technique, Técnica de Modelado de Objetos). Más tarde se les unió Ivar Jacobson, creador del método OOSE (Object-Oriented Software Engineering, Ingeniería de Software

Orientada a Objetos). En respuesta a una petición de OMG (Object Management Group, asociación para fijar los estándares de la industria) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos, en 1997 propusieron el UML como candidato [30].

Teniendo en cuenta las características del problema a resolver y el producto a desarrollar se decide utilizar UML ya que es de suma comodidad para el trabajo con lenguajes orientados a objetos el cual se utiliza en el proceso de implementación del producto.

### 1.10 Herramientas CASE.

Son el conjunto de herramientas que proporciona al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería [31]. Al igual que las herramientas de ingeniería y de diseño asistidos por computadora que utilizan los ingenieros de otras disciplinas, ayudan a garantizar que la calidad se diseñe antes de llegar a construir el producto. Entre los principales objetivos que se buscan al utilizar una herramienta CASE están:

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Reducir el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto
- Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

### 1.10.1 Rational Rose.

Rational Rose Enterprise Edition es una herramienta CASE desarrollada por Rational Corporation basada en el Lenguaje Unificado de Modelación (UML), que permite crear los diagramas que se van generando durante el proceso de Ingeniería en el Desarrollo del Software.

Rational Rose brinda muchas facilidades en la generación de la documentación del software que se esté desarrollando, además de que posee un gran número de estereotipos predefinidos que facilitan el proceso de modelación del software. Esta herramienta permite completar una gran parte de las disciplinas (flujos fundamentales) de RUP tales como captura de requisitos, análisis y diseño, implementación, control de cambios y gestión de configuración.

Entre las características principales de Rational se pueden destacar:

- Admite como notaciones: UML, OMT y Booch.
- Permite desarrollo multiusuario.
- Genera documentación del sistema.
- Disponible en múltiples plataformas.

### 1.10.2 Visual Paradigm.

Visual Paradigm es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre ediciones.

Se decide utilizar esta herramienta CASE como medio para modelar el proyecto ya que brinda grandes facilidades entre las que se encuentran:

- Soporte de UML.
- Modelado colaborativo con CVS y Subversion (control de versiones).
- Interoperabilidad con modelos UML2.

- Ingeniería inversa - Código a modelo, código a diagrama.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Generador de informes.

## 1.11 Herramientas de desarrollo de software.

Las herramientas de desarrollo de software desempeñan un papel primordial en la creación de aplicaciones. A lo largo de la vida de un proyecto de desarrollo de software, el programador puede utilizar diversas tecnologías y herramientas, las cuales muchas veces son definidas por supervisores o seleccionada por expertos.

Para la elección de las herramientas a utilizar se realiza un estudio para seleccionar las más adecuadas con las características de desarrollo.

### 1.11.1 Herramientas de programación.

Las herramientas de programación, son aquellas que permiten realizar aplicativos, programas, rutinas, utilitarios y sistemas para que la parte física del computador u ordenador, funcione y pueda producir resultados.

Actualmente existen múltiples herramientas de programación en el mercado, tanto para analistas expertos como para analistas inexpertos. Las herramientas de programación más comunes del mercado, cuentan hoy día con programas de depuración o debugger, que son utilitarios que nos permiten detectar los posibles errores en tiempo de ejecución o corrida de rutinas y programas [32].

Para el desarrollo del mediador se van a analizar tres de estas herramientas de las cuales se seleccionara una teniendo en cuenta sus características y prestaciones, tratando de escoger la que más se adecue a las necesidades de desarrollo.

**Aptana.**

Es un entorno de desarrollo dirigido hacia las aplicaciones web escritas en Ajax/JavaScript. Está basado en Eclipse y lo podremos encontrar para las tres plataformas mayoritarias (Win, Mac y Linux), ya sea como plugin del mismo Eclipse. Aptana Studio es gratuito, de código abierto y multiplataforma. Además permite trabajar con diferentes lenguajes y tecnologías de programación web como HTML, DOM, JavaScript (Js) y CSS.

Las características de este Entorno de Desarrollo Integrado (IDE) son similares a otros más generales: gestión de proyectos, vista outline y vista previa, autocompletado, macros, gestión de documentación, etc. Soporta las librerías más populares: Prototype, Scriptaculous, Dojo, MochiKit, Yahoo UI, Aflax, JQuery y Rico.

**Eclipse.**

Es un IDE desarrollado originalmente por IBM[11] como el sucesor de su familia de herramientas para VisualAge. Dispone de un editor de texto con resaltado de sintaxis. La compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, tests, etc., y refactorización. Asimismo, a través de "plugins" libremente disponibles es posible añadir control de versiones con Subversion e integración con Hibernate.

**Netbeans IDE.**

El IDE NetBeans está escrito en Java - pero puede servir para cualquier otro lenguaje de programación y existen además un número importante de módulos para extenderlo; además de ser un producto libre y gratuito sin restricciones de uso.

NetBeans es un producto de código abierto. Con su uso se obtienen todas las herramientas necesarias para crear profesionales de escritorio, de empresa, web y aplicaciones móviles con el lenguaje Java, C / C++, e incluso lenguajes dinámicos como PHP, Java Script, Groovy, y Ruby. Es fácil de instalar y utilizar, además, se ejecuta en muchas plataformas, incluyendo Windows, Linux, Mac OS X y Solaris [33].

Este IDE es el más recomendable para el desarrollo del software mediador pues contempla el framework y lenguaje a utilizar para la implementación permitiendo el completamiento de código y la integración con el entorno de desarrollo. Además es muy fácil de usar para la programación web y orientada a objetos.

### 1.11.2 Servidor web.

Un servidor web o servidor HTTP[12] es un programa informático que procesa una aplicación del lado del servidor realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente. Generalmente se utiliza el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI[13]. El término también se emplea para referirse al ordenador que ejecuta el programa [34].

WEBrick es una librería para servidores HTTP escrita por TAKAHASHI Masayoshi y GOTOU Yuuzou, además de los continuos parches aplicados por la inmensa comunidad de usuarios y desarrolladores de Ruby. Comenzó con un artículo titulado “Internet Programming with Ruby” en una revista de ingeniería de redes japonesa “OpenDesign”'. Y ahora, es parte de las librerías estándar desde Ruby 1.8.

En el paradigma de las aplicaciones Web, WEBrick es de un nivel bastante bajo. Sin embargo se usa bastante a menudo como servidor inicial para el entorno de desarrollo o de pruebas de las aplicaciones de Ruby. Esto es debido a que su configuración es muy sencilla (un fichero para cada tecnología externa a aplicar y un solo fichero de configuración interna). En general es fiable y la velocidad de servicio es aceptable. Es por esto que normalmente se utiliza este servidor para desarrollo y pruebas de las aplicaciones [35].

Se utiliza WEBrick como servidor para el desarrollo de la aplicación debido a las siguientes razones:

- Es el servidor más recomendado para desarrollo y pruebas de aplicaciones Web en Ruby.
- Es totalmente gratuito.
- Es open source.
- Su configuración es sencilla
- La mayoría de los IDEs de desarrollo con Ruby imponen este servidor por defecto al crear proyectos.

**Conclusiones del capítulo 1.**

- El estudio de los sistemas relacionados con la búsqueda en los repositorios permite concluir que presentan funcionalidades que resuelven el problema para los que fueron creados y que por sí solos no brindan solución a la problemática planteada en la presente investigación. Se muestran sus características, que sirven para determinar cuáles son las funciones principales que debe poseer un sistema de búsquedas para la solución.

- Se detallan las principales características de herramientas de desarrollo que son previamente condicionadas, como Ruby, Rails y PostgreSQL, para lograr una mejor comprensión y utilización en la Fase de Implementación del sistema.
- Al realizar un análisis sobre las metodologías y herramientas existentes en el mundo, se puede hacer la selección de las que más se adecuen a las características del entorno de desarrollo; siendo propuesta como metodología XP, que permite gestionar los cambios de requisitos de forma rápida y es muy adecuado para proyectos pequeños. Para el modelado se utiliza UML, lenguaje gráfico para visualizar, especificar, construir y documentar algunas de las partes que comprende el desarrollo del proyecto. Se propone como herramienta CASE el Visual Paradigm que soporta UML y es multiplataforma. El Netbeans es utilizado como IDE de desarrollo siendo muy fácil de usar para la programación web.

# CAPÍTULO 2
# EXPLORACIÓN, PLANIFICACIÓN Y DISEÑO.

**Introducción.**

Este capítulo está dedicado a valorar las características del sistema a desarrollar. Se realiza una descripción de la situación actual en la universidad donde surge la necesidad de optimizar el proceso de búsquedas dentro de un repositorio de activos reutilizables de software. Se representan conceptos y términos, los que facilitan la comprensión de los requerimientos software a desarrollar. Se describe el modelo de dominio, las historias de usuario así como el procedimiento a ejecutar por el usuario para el uso de las funcionalidades de la herramienta. Además, se explican los patrones de diseño y los patrones de arquitectura utilizados para el desarrollo.

## 2.1 Modelo de dominio.

Dada la propuesta de la creación del mediador para la búsqueda dentro de un repositorio de activos reutilizables se propone la definición de un modelo de dominio o modelo conceptual el cual muestra los principales conceptos a utilizar en el desarrollo de este complemento web.

Para lograr una mayor comprensión del dominio del problema, se crea el artefacto modelo conceptual o de dominio, que no es más que una representación visual de los conceptos u objetos del mundo real que son significativos para el problema o el área que se analiza; representando las clases conceptuales, no los componentes de software. Puede verse como un modelo que comunica a los interesados, cuáles son los términos importantes y cómo se relacionan entre sí.

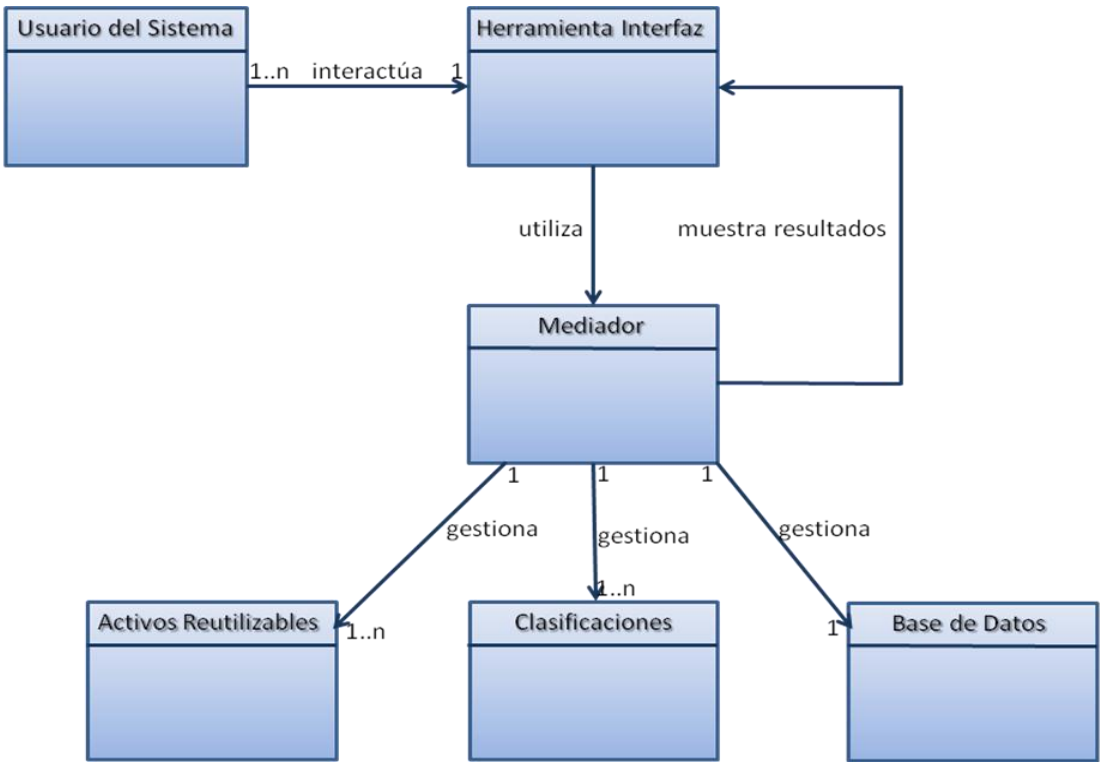



***Ilustración 4 Modelo de dominio.***

## 2.2 Fase de Exploración.

La metodología de desarrollo XP comienza con su fase de exploración. Durante esta fase se realiza el proceso de identificación de las Historias de Usuario (HU o UH, del inglés User Histories). Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología [29].

### 2.2.1 Historias de Usuarios.

Las HU son la forma en que se especifican en XP los requisitos del sistema. Estas se escriben desde la perspectiva del cliente, aunque los desarrolladores pueden brindar también su ayuda en la identificación de las mismas [36]. El contenido de estas debe ser concreto y sencillo [37].

El tratamiento de las HU es muy dinámico y flexible, en cualquier momento las historias de usuario pueden eliminarse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada HU es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [38].

Respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. En muchos casos sólo se propone utilizar un nombre y una descripción [39] o sólo una descripción [38], añadiendo quizás una estimación de esfuerzo en días [40]. Beck en su libro [41] presenta un ejemplo de ficha (customer story and task card) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado, cosas por terminar y comentarios.

Los contenidos de las fichas de las HU quedan estructurados de la siguiente forma:

**Número**: A cada HU se le asigna un número para facilitar su identificación por parte del equipo de desarrollo.

**Nombre**: Nombre descriptivo de la HU.

**Usuario**: Nombre de quien redacta la HU.

**Riesgo en desarrollo:** Grado de complejidad que le asigna el equipo de desarrollo a la HU luego de analizarla. (Alto, Medio o Bajo).

**Puntos estimados**: Unidades de tiempo estimadas por el equipo de desarrollo para darle cumplimiento a la HU. Una unidad de tiempo equivale a una semana de trabajo de 40 horas.

**Iteración asignada**: Número de la iteración en la cual será implementada la Historia de Usuario.

**Cómo probarlo**: Descripción a alto nivel de la forma en que se demuestra esta historia al final de cada iteración.

**Descripción**: Descripción simple sobre lo que debe hacer la funcionalidad en cuestión.

Durante la fase de exploración se identificaron cinco Historias de Usuario, las cuales se muestran a continuación.

**HU1**: Gestionar activos reutilizables de software.

**HU2**: Buscar activos reutilizables de software.

**HU3**: Mostrar la nube de etiquetas.

**HU4**: Listar activos reutilizables de software.

**HU5**: Buscar activos relacionados.

| **Historia de Usuario** | |
|---|---|
| **Número:** 1 | **Nombre**: Gestionar activos reutilizables de software |
| **Usuario:** Marcel Bauta | |
| **Riesgo en Desarrollo:** Alto | |
| **Puntos Estimados:** 5 | **Iteración Asignada:** 1 |
| **Descripción:** El usuario es el encargado de realizar la gestión de los activos reutilizables de software mediante la creación, modificación, o eliminación de los mismos. Desde la página principal de la aplicación se puede acceder a la opción de crear un nuevo activo y desde las demás interfaces de búsquedas se pueden modificar, eliminar o descargar. | |
| **Cómo probarlo:** Mediante las interfaces correspondientes, insertar un nuevo activo con nombre *activo1* y url, *http://activo1.com*, comprobar que se insertó correctamente mediante la funcionalidad mostrar activo, luego editar ese activo cambiándole el nombre por *activo2* y comprobar que los cambios fueron hechos correctamente. Para finalizar eliminar el activo y chequear que no se encuentre en la base de datos. | |

| **Historia de Usuario** | |
|---|---|
| **Número:** 2 | **Nombre**: Buscar activos reutilizables de software |

| | |
|---|---|
| **Usuario:** Marcel Bauta | |
| **Riesgo en Desarrollo:** Alto | |
| **Puntos Estimados:** 4 | **Iteración Asignada:** 1 |
| **Descripción:** El usuario es el encargado de realizar las acciones pertinentes sobre el repositorio como la búsqueda de un activo introduciendo el nombre de las etiquetas, luego el sistema lista los activos que se corresponden con el(los) criterio(s) de búsqueda y por último el usuario descarga, modifica o elimina el activo localizado. También se pueden introducir estas etiquetas a la búsqueda mediante la navegación por la nube. | |
| **Cómo probarlo:** Teniendo activos en la base de datos etiquetados con “linux” y “windows”, mediante las interfaces correspondientes, realizar una búsqueda de los activos que tengan con estas etiquetas. Comprobar que la lista generada se corresponda con los datos de los activos de la BD. | |

| **Historia de Usuario** | |
|---|---|
| **Número:** 3 | **Nombre**: Mostrar la nube de etiquetas |
| **Usuario:** Marcel Bauta | |
| **Riesgo en Desarrollo:** Alto | |
| **Puntos Estimados:** 3.5 | **Iteración Asignada:** 2 |
| **Descripción:** El usuario es el encargado de inicializar la acción al seleccionar la opción de generar la nube en la interfaz principal de la aplicación. Seguidamente la nube de contextos y sus respectivas etiquetas es mostrada, permitiendo la selección de ellas y mostrando una lista de todos los activos que se correspondan con dicha etiqueta. | |
| **Cómo probarlo:** Comprobar que en la base de datos estén los contextos “SO” con las etiquetas “linux” | |

| |
|---|
| y “windows” y “Environment” con las etiquetas “web” y “desktop. Seguidamente mediante la interfaz correspondiente, la nube con las etiquetas “web, desktop, linux y windows” debe ser mostrada. |

| **Historia de Usuario** | |
|---|---|
| **Número:** 4 | **Nombre**: Listar activos reutilizables de software |
| **Usuario:** Marcel Bauta | |
| **Riesgo en Desarrollo: M**edio | |
| **Puntos Estimados:** 3.5 | **Iteración Asignada:** 2 |
| **Descripción:** El usuario es el encargado inicializar la acción al seleccionar la opción de listar todos los activos en la interfaz principal de la aplicación. Seguidamente una lista o tabla con todos los activos presentes en la base de datos es mostrada, permitiendo la modificación, eliminación o descarga de ellos. | |
| **Cómo probarlo:** Comprobar que en la base de datos existan los activos 1 y 2 con sus respectivos atributos, seguidamente mediante la interfaz correspondiente, estos deben ser mostrados en una tabla. | |

| **Historia de Usuario** | |
|---|---|
| **Número:** 5 | **Nombre**: Buscar activos relacionados |
| **Usuario:** Marcel Bauta. | |
| **Riesgo en Desarrollo: M**edio | |
| **Puntos Estimados:** 3 | **Iteración Asignada:** 3 |

| |
|---|
| **Descripción:** El usuario es el encargado inicializar la acción al seleccionar la opción de buscar los activos relacionados en la interfaz principal de la aplicación. Seguidamente una lista con todos los activos presentes en la base de datos es mostrada, permitiendo la selección de uno de ellos y mostrando seguidamente todos los activos que posean etiquetas similares en su total o parcial correspondencia. |
| **Cómo probarlo:** Mediante las interfaces correspondientes, en la base de datos en blanco, insertar tres nuevos activos, uno con nombre *activo1* y url *http://activo1.com*, y la etiqueta “linux”; otro con nombre *activo2* y url *http://activo2.com*, y la etiqueta “web”; y por último uno con nombre *activo3* y url *http://activo3.com*, y las etiquetas “linux” y “web”. Para finalizar al buscar los activos relacionados con el *activo3* se deben mostrar una lista con el *activo1* y el *activo2*. |

### 2.3 Fase de Planificación.

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

Las estimaciones de esfuerzo asociadas a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la “velocidad” de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración [29].

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario

seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación [29].

### 2.3.1 Estimación del esfuerzo por Historia de Usuario.

Durante la fase de planificación se realiza la estimación del esfuerzo que costará implementar cada historia de usuario. Esto se expresa utilizando como medida el punto. Un punto se considera como una semana ideal de trabajo, donde los miembros de los equipos de desarrollo trabajan el tiempo planeado sin ningún tipo de interrupción. Esta estimación incluye todo el esfuerzo asociado a la implementación de la historia de usuario, por ejemplo, las pruebas unitarias, la integración y refactorización del código, y la preparación y ejecución de las pruebas de aceptación.

A continuación se muestra mediante una tabla la planificación de la estimación del esfuerzo para las diferentes historias de usuario:

| No | Historia de Usuario | Puntos de Estimación |
|---|---|---|
| **1** | Gestionar activos reutilizables de software. | 5 |
| **2** | Buscar activos reutilizables de software. | 4 |
| **3** | Mostrar la nube de etiquetas. | 3.5 |
| **4** | Listar activos reutilizables de software. | 3.5 |
| **5** | Buscar activos relacionados | 3 |

***Tabla 1 Estimación del esfuerzo por HU.***

El tiempo total estimado para el desarrollo del sistema es de 19 puntos, que equivalen a 19 semanas ideales de trabajo, con jornadas laborales de 8h.

### 2.3.2 Plan de iteraciones.

Una vez identificadas las HU y estimado el esfuerzo propuesto para el desarrollo del sistema de cada una de estas historias, se procede a la planificación de la etapa de implementación del sistema [36]. Teniendo en cuenta la prioridad que tiene una determinada historia de usuario en el desarrollo de la herramienta se

decide en que iteración será implementada. Las historias de usuario que cuentan con mayor importancia por ser funcionalidades indispensables para la aplicación deben ser implementadas en las primeras iteraciones del ciclo de desarrollo. A continuación se decide realizar el mediador en tres iteraciones, las cuales son:

**Iteración 1.**

Durante el transcurso de esta iteración se crea la base de la arquitectura del sistema logrando cierta funcionalidad. Al final se cuenta con una primera versión de prueba, la cual es testeada con el objetivo de obtener una retroalimentación para el grupo de trabajo y con las funcionalidades descritas en las HU No 1 y 2.

**Iteración 2.**

Con la culminación de esta iteración se tiene implementadas las peticiones del cliente descritas en la historia de usuario No 3 y 4. Se cuenta con una versión de prueba de las funcionalidades antes mencionadas, además de las implementadas en la iteración anterior.

**Iteración 3.**

En esta iteración se implementan las funcionalidades descritas en la historia de usuario No 5. Como resultado se obtiene la versión 1.0 del producto final. A partir de este momento el sistema es puesto a prueba por un período de tiempo para evaluar el desempeño del mismo.

### 2.3.3 Plan de duración de las iteraciones.

En el ciclo de vida de un proyecto regido por la Metodología XP se crea el plan de duración de cada una de las iteraciones, en este caso se hace para el único equipo de desarrollo con el cual se cuenta. Este plan tiene como finalidad mostrar el tiempo de duración de cada iteración, así como el orden en que se implementan las HU en cada una de las mismas.

| No | Iteración | Orden de las HU a implementar | Duración total de la iteración |
|---|---|---|---|
| **1** | Iteración 1 | Gestionar activos reutilizables de software<br>Buscar activos reutilizables de software | 9 semanas |
| **2** | Iteración 2 | Mostrar la nube de etiquetas<br>Listar activos reutilizables de software | 7 semanas |
| **3** | Iteración 3 | Buscar activos relacionados | 3 semanas |

***Tabla 2 Plan de duración de las iteraciones.***

### 2.3.4 Plan de entregas.

A continuación se presenta el Plan de entregas ideado para la fase de implementación, donde se hace una propuesta de la fecha aproximada en que se liberan las versiones (releases) del sistema al finalizar cada iteración.

| Entregable | Final 1ra iteración<br>4ta semana de marzo | Final 2da iteración<br>2da semana de mayo | Final 3ra iteración<br>5ta semana de mayo |
|---|---|---|---|
| BUSCADOR DE ACTIVOS REUTILIZABLES | v0.1 | v0.2 | v1.0 |

***Tabla 3 Plan de entregas.***

## 2.4 Diseño del sistema.

El diseño adecuado para el software es aquel que supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación del equipo de desarrollo y tiene el menor número posible de clases y métodos [41].

Para el diseño de las aplicaciones, la metodología XP no requiere la representación del sistema mediante diagramas de clases, aunque su inclusión dentro de los artefactos generados en el ciclo de vida del proyecto puede servir para una mejor comprensión del problema (Anexo 1). En su lugar se usan otras técnicas como las tarjetas CRC (clase, responsabilidad y colaboración) como una extensión informal a UML las cuales surgen de las sesiones CRC que se explican más adelante.

### 2.4.1 Patrones de Diseño.

Los patrones de diseño son descripciones de clases cuyas instancias colaboran entre sí. Cada patrón es adecuado para ser adaptado a un cierto tipo de problema. Representa un esquema o microarquitectura que supone una solución a problemas (dominios de aplicación) semejantes; una estructura común que tienen aplicaciones semejantes. [44]

Los patrones de diseño brindan una solución generalmente ya probada y documentada a problemas que se dan durante el proceso de desarrollo de software. Emplean un conjunto de buenas prácticas que facilitan el trabajo, definen una estructura de clases que da respuesta a uno o varios problemas en particular y presentan la ventaja de que son fáciles de comprender, además de que no dependen del lenguaje, haciéndolos genéricos. Lo complejo es cuando se tiene que decidir cuál usar, pues presentan diferentes soluciones, ya sea a través del empleo de uno u otro, o la combinación de varios. De ahí la importancia de conocer y estudiar los diferentes patrones que existen para poder determinar su uso.

A continuación se relacionan los patrones de diseño empleados y su aplicación en la realización de los diagramas de clases del diseño para su posterior implementación.

**Relación entre los patrones y el framework Rails.**

Los frameworks utilizan un variado número de patrones de diseño, ya que así logran soportar aplicaciones de más alto nivel y que reutilicen una mayor cantidad de código. “Los patrones ayudan a hacer la arquitectura de los frameworks más adecuada para muchas y diferentes aplicaciones sin necesidad de rediseño” [45]. Estos patrones son usados de forma implícita cuando se trabaja con el marco de trabajo Rails, que los define e implementa.

**Patrón Active Record.**

Es un patrón de diseño para el manejo de la lógica del acceso a la BD. En este, se representan las filas de las tablas de una BD en forma de objetos, de esta forma el acceso a datos se mantiene de manera

uniforme a través de la aplicación. Active Record abstrae la manera de manipular los objetos mediante métodos internamente definidos que generan las consultas SQL necesarias para conectarse e interactuar con la BD.

En el sistema desarrollado, este patrón tiene un uso primordial pues los valores de los atributos de los modelos como los nombres, las direcciones y las etiquetas de los activos, se guardan en tablas de la base de datos relacional; siendo Active Record el encargado de gestionar las consultas realizadas. La implementación de Active Record en *Rails* incluye declaraciones como *has_many*, *belongs_to* y *has_and_belongs_to_many* que expresan los tipos de relaciones entre las tablas *Actives*, *Tags*, *Contexts* del modelo de datos.

**Patrones GRASP (del inglés General Responsibility Assignment Software Patterns).**

Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones [30].

***Experto***: Asignar una responsabilidad al más competente en información, la clase cuenta con la información necesaria para cumplir la responsabilidad. Es el principio básico de asignación de responsabilidades que suele utilizarse en el diseño orientado a objetos [30].

Se utiliza de manera implícita en todo el marco de trabajo, pues las clases principales están diseñadas para que tengan bien definida sus responsabilidades. Cada uno de los componentes creados para desarrollar la aplicación tiene las bases para el cumplimiento de este patrón, ya sean en los modelos, las vistas o las clases controladoras, dejando una parte de la responsabilidad al desarrollador. Por convención se define que en las Vistas se desarrolla lo que será mostrado al usuario, en el Modelo, por ejemplo es donde se garantiza las validaciones y en los Controladores, donde se definen las funcionalidades.

***Controlador***: Asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. Asigna la responsabilidad del manejo de mensajes de los eventos de un sistema a una clase. Garantiza que los procesos sean manejados por la capa Controladora y no por la capa de presentación [30].

La utilización de este patrón se evidencia en la no existencia de llamadas a la base de datos desde las vistas implementadas, sino a través de la clase controladora de cada entidad definida, por ejemplo *ActivesController*.

***Alta Cohesión***: Asignar una responsabilidad de modo que la unión se mantenga a gran escala. Asignar a las clases responsabilidades que trabajen sobre una misma área de aplicación y que no tengan mucha complejidad [30].

*Rails* mediante mecanismos internos garantiza que se mantenga una alta cohesión entre los modelos, las vistas y las clases controladoras, en la implementación se ha respetado este patrón por ejemplo, no teniendo varias funciones para acceder a los datos, o permitiendo llamadas desde varios controladores a una misma vista.

***Bajo Acoplamiento***: Asignar una responsabilidad para mantener un engranaje pobre. Es un principio que se debe recordar durante las decisiones de diseño. Soporta el diseño de clases más independientes. Asignar las responsabilidades de forma tal que las clases se comuniquen con el menor número de clases que sea posible [30].

Las clases de las vistas, los modelos del dominio y las controladoras son totalmente independientes unas de otras, o sea, existe la menor relación posible entre estas clases debido a la separación en capas que hace el patrón arquitectónico.

### 2.4.2 Patrones de arquitectura.

Los patrones arquitectónicos, o patrones de arquitectura, son patrones de diseño que ofrecen soluciones a problemas de arquitectura en la ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

El Patrón Arquitectónico es el nivel en el cual la arquitectura de software:

- Define la estructura básica de un sistema, pudiendo estar relacionado con otros patrones
- Representa una plantilla de construcción que provee un conjunto de subsistemas aportando las normas para su organización.
- Ejemplos: Capas, MVC, Tuberías y Filtros, Pizarra, etc.

**Patrón Modelo-Vista-Controlador (MVC).**

El MVC es el patrón de arquitectura en el que se basa *Rails* y propone para hacer el proceso de desarrollo, por lo que la viabilidad de utilizar otro para la implementación es incompatible con el presente marco de trabajo. Este es el encargado de separar la lógica de negocio de la interfaz del usuario y es el más utilizado en aplicaciones Web, ya que facilita la funcionalidad, mantenibilidad y escalabilidad del sistema, de forma simple y sencilla [42].

El Modelo, es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, sin tomar en cuenta ni la forma en la que esa información va a ser mostrada ni los mecanismos que hacen que esos datos estén dentro del modelo, es decir, sin tener relación con ninguna otra entidad dentro de la aplicación [43].

Las Vistas, son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo; así por ejemplo, se puede tener una vista mostrando la hora del sistema como un reloj analógico y otra vista mostrando la misma información como un reloj digital [43].

El Controlador, es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador [43].

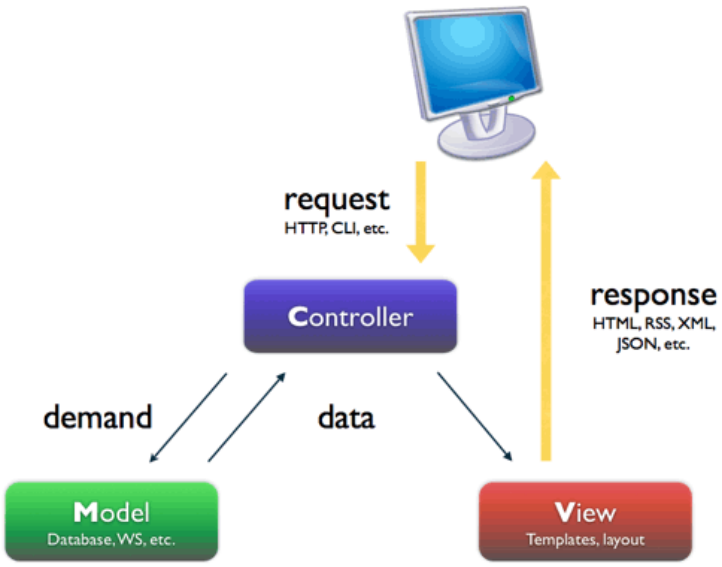



***Ilustración 5 Patrón MVC.***

### 2.4.3 Tarjetas Clase, Responsabilidades y Colaboración (CRC).

Las tarjetas CRC son una herramienta de reflexión en el diseño de software orientado a objetos. Fueron propuestas por Ward Cunningham y Kent Beck. Se utilizan normalmente cuando primero se determinan las clases que se necesitan y cómo van a interactuar y después se implementa la solución.

La técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis dirigidos por la responsabilidad. Las clases se examinan, se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y con las que necesitan colaborar para completar sus responsabilidades. Es importante resaltar que esta tarea es permanente durante la vida del proyecto partiendo de un diseño inicial que va siendo corregido y mejorado en el transcurso de este.

Una sesión CRC es una técnica donde los miembros del equipo se reúnen en una habitación con una mesa de trabajo de tamaño preferiblemente mediano o grande, las tarjetas CRC en blanco son puestas encima y estos son los encargado mediante la conversación y retroalimentación de ideas, escribir los datos pertenecientes a los nombres de las clases, sus responsabilidades y colaboraciones. Esta técnica permite la total interacción de los miembros del equipo a la hora de resolver un problema y es un tipo de tormenta de ideas. Durante la sesión las tarjetas son movidas de un lado hacia otro y las

responsabilidades y colaboraciones pueden cambiar también, lo que se desea lograr es que todos los participantes aporten buenas ideas para el desarrollo del sistema.

La parte posterior de las tarjetas CRC es usada para dar una mayor explicación sobre la clase y otras notas capturadas durante la sesión CRC. En algunos casos se incluyen los atributos de las clases.

Características que presentan las tarjetas CRC:

- Es una técnica para la representación de sistemas orientados a objetos, para pensar en objetos.
- Son un puente de comunicación entre diferentes participantes.
- Principales desventajas: lentitud y roces.
- Se recomienda un grupo de trabajo con representantes de las distintas partes.
- Permite ver las clases como algo más que un repositorio de datos.
- Posibilita conocer el comportamiento de cada clase en un alto nivel.

Las partes que componen una tarjeta CRC son las siguientes:

- Clase: Nombre de la clase con que se está modelando.
- Súper Clase: Nombre de la clase padre en la herencia.
- Sub Clase(s): Nombre de las clases hijas en la herencia.
- Responsabilidades: Descripción de alto nivel del propósito de la clase.
- Colaboración: Indica con cuáles otras clases se requiere relación para cumplir con las responsabilidades.

Estas tarjetas CRC se hacen con el objetivo de identificar jerarquías de generalización/especificación, o jerarquías de agregación entre las clases, de manera que ayuda al refinamiento de estas. Se definen además con la finalidad de obtener un diseño simple y no incurrir en la implementación de características que no son necesarias. A continuación se presentan las tarjetas CRC pertenecientes al mediador BuscActivo, diseñadas con los principales campos que las componen.

| **Tarjeta CRC** | |
|---|---|
| **Nombre de la Clase: ApplicationController** | |
| Responsabilidades | Colaboración |
| render403()<br>render404()<br>render_error()<br>use_layout() | ActivesController<br>SearchController |

| **Tarjeta CRC** | |
|---|---|
| **Nombre de la Clase: ActivesController** | |
| Responsabilidades | Colaboración |
| index()<br>show()<br>nube()<br>new()<br>edit()<br>create()<br>update()<br>destroy()<br>**private** find_active() | Tagging<br>Actives |

| Tarjeta CRC | |
|---|---|
| **Nombre de la Clase: SearchController** | |
| Responsabilidades | Colaboración |
| index()<br>tagged_with()<br>related_tags()<br>related_active() | Tagging<br>Actives |

| Tarjeta CRC | |
|---|---|
| **Nombre de la Clase: Tagging** | |
| Responsabilidades | Colaboración |
| tag_counts_on()<br>tag_types()<br>tagged_with()<br>tag_list()<br>is_taggable()<br>find_related()<br>is_tagger() | Actives<br>ActivesController<br>SearchController |

| **Tarjeta CRC** | |
|---|---|
| **Nombre de la Clase: Actives** | |
| Responsabilidades | Colaboración |
| | Contexts<br>Tags<br>Taggings |

| **Tarjeta CRC** | |
|---|---|
| **Nombre de la Clase: Contexts** | |
| Responsabilidades | Colaboración |
| | Actives<br>Tags |

| **Tarjeta CRC** | |
|---|---|
| **Nombre de la Clase: Tags** | |
| Responsabilidades | Colaboración |
| | Actives<br>Contexts |

**Diagrama de Despliegue.**

El diagrama de despliegue describe la arquitectura física del sistema durante la ejecución en términos de procesadores, dispositivos y componentes de software. Describe también la topología del sistema, es decir la estructura de los elementos de hardware y software que ejecuta cada uno de ellos.

Partes que lo componen:

*Nodos*: son objetos físicos que existen en tiempo de ejecución y representan algún tipo de recurso computacional, ej.

- Computadores con procesadores
- Impresoras
- Lectoras de códigos de barras

*Dispositivos*: Los dispositivos del sistema también se representan como nodos. Generalmente se usan estereotipos para identificar el tipo de dispositivo.

*Nombre de dispositivo*: descripción de la capacidad que el dispositivo provee al sistema.

*Nombre del procesador*: descripción de la funcionalidad y capacidad del nodo.

En el Anexo 2 se muestra el diagrama de despliegue del sistema.

**Conclusiones del capítulo 2.**

Durante la elaboración de este capítulo, el estudio de las fases de Exploración y Planificación propias de la metodología de desarrollo utilizada para la implementación de la aplicación, permite arribar a las siguientes conclusiones:

- Se asume una implementación por etapas la cual fue concebida y debidamente detallada. Durante la Fase de Exploración se describen las Historias de Usuarios en correspondencia con cada uno de los requisitos funcionales identificados y se hace una descripción de todos los artefactos generados mediante las HU para realizar una implementación con el mínimo de esfuerzo por parte de los programadores.
- Durante la Fase de Planificación se realiza la estimación del esfuerzo por cada HU; se realiza el plan de iteraciones y el plan de duración de cada iteración y se define un plan de entregas con las propuestas de liberación de las versiones de la aplicación, que permiten tener un estimado de tiempo para el desarrollo.

# CAPÍTULO 3
# IMPLEMENTACIÓN Y PRUEBAS.

**Introducción.**

En el presente capítulo se describen las fases de implementación y pruebas. Se detallan las tareas de programación pertenecientes a las iteraciones llevadas a cabo durante la etapa de codificación del sistema. Además se hace alusión a la fase de prueba, propia de la metodología de desarrollo utilizada para el desarrollo de la aplicación. También se detallan las pruebas unitarias y las pruebas de aceptación realizadas sobre el sistema.

## 3.1 Fase de implementación.

La implementación o codificación es el flujo de trabajo donde se producen los componentes del sistema: ejecutables, ficheros de código fuente, scripts, entre otros. Tiene como objetivo principal desarrollar la arquitectura y el sistema como un todo, así como definir la organización del código. La metodología XP plantea que la implementación de un software debe realizarse de forma iterativa, obteniendo al culminar cada iteración un producto funcional que debe ser probado y mostrado al cliente para incrementar la visión de los desarrolladores con la opinión de éste.

En esta fase se genera todo el código fuente necesario para satisfacer las HU definidas para la solución y se describen todas las tareas realizadas en cada iteración. Al inicio de cada HU, se lleva a cabo una revisión del plan de iteraciones y se modifica de ser necesario. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador o grupo de programadores como responsables. Estas tareas, pueden escribirse utilizando un lenguaje técnico y no necesariamente deben ser entendibles para el cliente.

Para la implementación del software mediador se determinaron en la fase de planificación tres iteraciones de desarrollo. A continuación se hace referencia a los principales aspectos de las tareas de programación realizadas en las distintas iteraciones.

| No | Historia de Usuario | Iteración | Tareas de programación | | |
|---|---|---|---|---|---|
| | | | No | Nombre | Puntos estimados |
| **1** | Gestionar activos reutilizables de software. | 1 | 1 | Crear estructura inicial del mediador usando las herramientas propuestas. | 1 |
| | | | 2 | Diseñar interfaz gráfica de usuario para la HU1. | 0.5 |
| | | | 3 | Implementar las opciones de crear, modificar, eliminar o descargar activos reutilizables. | 2.5 |
| | | | 4 | Escribir pruebas para la HU1. | 1 |
| **2** | Buscar activos reutilizables de software. | 1 | 5 | Diseñar interfaz gráfica de usuario para la HU2. | 0.5 |
| | | | 6 | Implementar las funcionalidades de búsqueda por etiquetas. | 2.5 |
| | | | 7 | Escribir pruebas para la HU2. | 1 |
| **3** | Mostrar la nube de etiquetas. | 2 | 8 | Diseñar interfaz gráfica de usuario para la HU3. | 0.5 |
| | | | 9 | Implementar las funcionalidades de la nube de etiquetas. | 2 |
| | | | 10 | Escribir pruebas para la HU3. | 1 |
| **4** | Listar activos reutilizables de software. | 2 | 11 | Diseñar interfaz gráfica de usuario para la HU4. | 0.5 |
| | | | 12 | Implementar la funcionalidad de listar todos los activos reutilizables. | 2 |

| | | | | | |
|---|---|---|---|---|---|
| | | | 13 | Escribir pruebas para la HU4. | 1 |
| **5** | Buscar activos relacionados. | 3 | 14 | Diseñar interfaz gráfica de usuario para la HU5. | 0.5 |
| | | | 15 | Implementar la funcionalidad de buscar todos los activos relacionados. | 1.5 |
| | | | 16 | Escribir pruebas para la HU5. | 1 |

***Tabla 4 Tareas de programación.***

### 3.1.1 Iteración 1.

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 1 | **Número de la HU:** 1 |
| **Nombre de la tarea:** Crear estructura inicial del mediador usando las herramientas propuestas. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a crear las condiciones de trabajo para la implementación del mediador. Se debe crear la base de datos en PostgreSQL, e integrar el framework Rails con el IDE Netbeans. Luego comprobar que se ejecute correctamente, para crear la estructura del mediador. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 2 | **Número de la HU:** 1 |
| **Nombre de la tarea:** Diseñar interfaz gráfica de usuario para la HU1. | |

| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 0.5 |
|---|---|
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a diseñar los campos y formularios necesarios en la implementación de la HU1. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 3 | **Número de la HU:** 1 |
| **Nombre de la tarea:** Implementar las opciones de crear, modificar, eliminar o descargar activos reutilizables. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 2.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a crear los atributos y métodos necesarios para la correcta gestión de los activos en la base de datos. Se deben generar los controladores, modelos y vistas asociados a las clases, así como la correcta validación de estos datos. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 4 | **Número de la HU:** 1 |
| **Nombre de la tarea:** Escribir pruebas a la HU1. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1 |
| **Programador(es) responsable(s):** Marcel Bauta. | |

| |
|---|
| **Descripción:** Tarea dirigida a escribir las pruebas de aceptación y unitarias de la HU1. |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 5 | **Número de la HU:** 2 |
| **Nombre de la tarea:** Diseñar interfaz gráfica de usuario para HU2. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 0.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a diseñar los campos y formularios necesarios en la implementación de la HU2. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 6 | **Número de la HU:** 2 |
| **Nombre de la tarea:** Implementar las funcionalidades de búsqueda por etiquetas. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 2.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a crear los atributos y métodos necesarios para la correcta gestión de la búsqueda de activos mediante etiquetas en la base de datos. Se deben generar los controladores, modelos y vistas asociados a las clases, así como la correcta validación de estos datos. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 7 | **Número de la HU:** 2 |
| **Nombre de la tarea:** Escribir pruebas a la HU2. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a escribir las pruebas de aceptación y unitarias de la HU2. | |

### 3.1.2 Iteración 2.

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 8 | **Número de la HU:** 3 |
| **Nombre de la tarea:** Diseñar interfaz gráfica de usuario para la HU3. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 0.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a diseñar los campos y formularios necesarios en la implementación de la HU3. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 9 | **Número de la HU:** 3 |
| **Nombre de la tarea:** Implementar las funcionalidades de la nube de etiquetas. | |

| | |
|---|---|
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 2 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a crear los atributos y métodos necesarios para la correcta gestión de la nube de etiquetas. Se deben generar los controladores, modelos y vistas asociados a las clases, así como la correcta validación de estos datos. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 10 | **Número de la HU:** 3 |
| **Nombre de la tarea:** Escribir pruebas a la HU3. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a escribir las pruebas de aceptación y unitarias de la HU3. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 11 | **Número de la HU:** 4 |
| **Nombre de la tarea:** Diseñar interfaz gráfica de usuario para HU4. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 0.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea para diseñar los campos y formularios necesarios en la implementación de la HU4 | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 12 | **Número de la HU:** 4 |
| **Nombre de la tarea:** Implementar la funcionalidad de listar todos los activos reutilizables. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 2 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a crear los atributos y métodos necesarios para la correcta gestión de la lista de activos que se encuentran en la base de datos. Se deben generar los controladores, modelos y vistas asociados a las clases, así como la correcta validación de estos datos. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 13 | **Número de la HU:** 4 |
| **Nombre de la tarea:** Escribir pruebas a la HU4. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a escribir las pruebas de aceptación y unitarias de la HU4. | |

### 3.1.3 Iteración 3.

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 14 | **Número de la HU:** 5 |

| | |
|---|---|
| **Nombre de la tarea:** Diseñar interfaz gráfica de usuario para HU5. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 0.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a diseñar los campos y formularios necesarios en la implementación de la HU4. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 15 | **Número de la HU:** 5 |
| **Nombre de la tarea:** Implementar la funcionalidad de buscar todos los activos relacionados. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1.5 |
| **Programador(es) responsable(s):** Marcel Bauta. | |
| **Descripción:** Tarea dirigida a crear los atributos y métodos necesarios para la correcta gestión de la búsqueda de activos relacionados que se encuentran en la base de datos. Se deben generar los controladores, modelos y vistas asociados a las clases, así como la correcta validación de estos datos. | |

| **Tarea de Programación** | |
|---|---|
| **Número de la tarea:** 16 | **Número de la HU:** 5 |
| **Nombre de la tarea:** Escribir pruebas a la HU5. | |
| **Tipo de tarea:** Desarrollo | **Puntos Estimados:** 1 |

| **Programador(es) responsable(s):** Marcel Bauta. |
| --- |
| **Descripción:** Tarea dirigida a escribir las pruebas de aceptación y unitarias de la HU5. |

### 3.2 Diseño de la base de datos del sistema.

Las bases de datos (BD) necesitan una definición de su estructura que permita almacenar datos, reconocer el contenido y recuperar la información. Su estructura tiene que ser desarrollada para la necesidad de las aplicaciones que la usarán, por lo que se diseñan a partir de los requisitos del proceso del negocio, que es la primera abstracción de la vista de la base de datos.

En el proceso y construcción de todo sistema informativo automatizado, el diseño de la BD ocupa un lugar importante, a tal punto que esta puede verse como un proceso relativamente independiente dentro del diseño del sistema y compuesto por una serie de etapas. Es por ello que resulta de interés el estudio de los problemas relacionados con el diseño de las bases de datos y la modelación de la información. A continuación se muestra el modelo físico de la base de datos utilizado en la construcción del sistema.

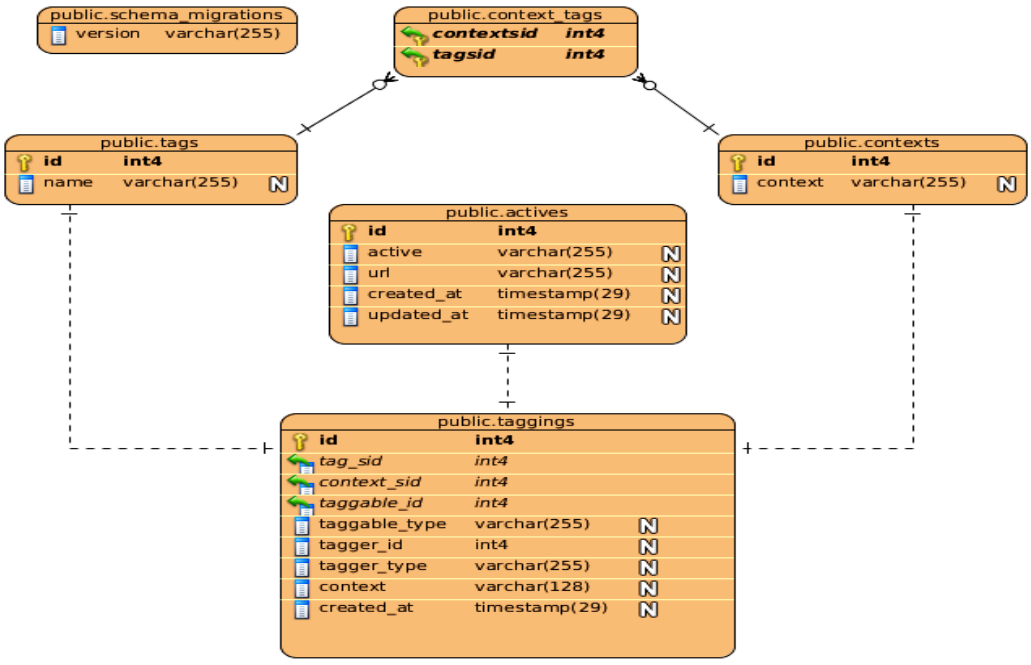



***Ilustración 6 Modelo físico de la base de datos.***

### 3.3 Estándar de codificación.

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código.

El usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Los estándares de codificación se definen por el equipo de desarrollo para lograr generalización en la programación del software. La generalización de aspectos tan simples como el trato de las mayúsculas, ayuda a eliminar conflictos de funcionalidades implementadas con nombres iguales y guían de forma clara el proceso de desarrollo. A continuación se muestran algunos de los ejemplos de estándares que se toman en cuenta para el desarrollo del mediador BuscActivo.

El código está delimitado por la forma de las etiquetas ruby estándares: **<%= %>**

La longitud utilizada para una línea de código es de 120 caracteres y se debe mantener por debajo de esta cifra siempre que sea posible, para tener un código más legible y contar con una detección más fácil de errores.

Los nombres de las clases se crean utilizando PascalCasing iniciando cada palabra con letra mayúscula y el resto en minúscula. Ej ActivesController.

Los nombres de las funciones contienen únicamente caracteres alfabéticos escritos en minúscula y los guiones bajos (_) se utilizan para separarlos en caso de que sean nombres compuestos. Ej new, show, tag_list, etc.

Los nombres de las variables contienen caracteres alfabéticos y los guiones bajos (_) se utilizan para separarlos en caso de que sean nombres compuestos. A continuación se muestran los distintos tipos de variables.

| Variable | Tipo |
|---|---|
| var | variable local |
| @var | variable de instancia |
| @@var | variable de clase |
| $var | variable global |
| VAR | constante |

***Tabla 5 Estándar de variables.***

### 3.4 Fase de pruebas.

Uno de los pilares fundamentales de XP es el proceso de pruebas, el cual anima a los desarrolladores a probar constantemente tanto como sea posible. Mediante esta filosofía se reduce el número de errores no detectados así como el tiempo entre la introducción de este en el sistema y su detección. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores a la hora de introducir cambios o modificaciones [46].

La metodología XP utiliza dos tipos de pruebas fundamentales, las pruebas unitarias, desarrolladas por los programadores, encargadas de verificar el código y las pruebas de aceptación, destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada. Por su alta importancia se realiza también una serie de pruebas funcionales al sistema para validar el correcto funcionamiento de sus clases controladoras y su relación con las vistas.

#### 3.4.1 Pruebas unitarias y funcionales.

Las pruebas unitarias son una de las piedras angulares de XP, todos los módulos deben pasarlas antes de ser liberados o publicados. Que todo código liberado pase correctamente las pruebas unitarias es lo que habilita que funcione la propiedad colectiva del código. En este sentido, el sistema y el conjunto de pruebas debe ser guardado junto con el código, para que pueda ser utilizado por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo [47].

Se denominan pruebas funcionales, a las pruebas de software que tienen por objetivo asegurar que los sistemas desarrollados, cumplan con las funciones específicas para los cuales han sido creados. A este tipo de pruebas se les denomina también pruebas de comportamiento o pruebas de caja negra, ya que los probadores o analistas de pruebas, no enfocan su atención a como se generan las respuestas del sistema, básicamente el enfoque se basa en el análisis de los datos de entrada y en los de salida.

Los sistemas que han pasado por pruebas unitarias tienen un menor tiempo de pruebas funcionales, este comportamiento es obvio, ya que las pruebas unitarias nos permiten encontrar los errores más evidentes y fáciles de corregir, en la etapa de pruebas funcionales el sistema debería estar bastante estable y con muy pocos errores críticos. Si un sistema llega a la etapa de pruebas funcionales con demasiados errores críticos y/o bloqueantes, se debería devolver el sistema a la etapa de pruebas unitarias ya que resulta muy poco productivo realizar pruebas funcionales con sistemas inestables, el avance es demasiado lento [48].

La automatización de pruebas unitarias y funcionales puede resultar compleja y solo se recomienda en algunas funcionalidades específicas, por ejemplo en las vistas que tendrán mayor uso, generalmente de ingreso de datos. La metodología XP propone la utilización de una herramienta que realice de forma automática la ejecución de las pruebas unitarias y funcionales. Rails utiliza el framework de pruebas Test::Unit para la implementación de estas.

**Pruebas unitarias a los modelos.**

Las pruebas unitarias en Rails son realizadas a los modelos creados por ActiveRecord. Luego de escribir el código para las validaciones de los datos, en lugar de probarlos manualmente insertando los datos, se utilizan este tipo de pruebas para confirmar que el sistema se comporte de la manera esperada, haciendo uso de las restricciones previamente escritas, llamando a los métodos y comparando los valores que retorna con lo que se espera que estos devuelvan. Siempre que se use un generador para crear un modelo ActiveRecord, Rails automáticamente crea una plantilla de prueba unitaria en el directorio *test/unit* del proyecto, en las que se programan las específicas del software. A continuación se presenta el código y los resultados de las pruebas unitarias hechas al sistema.

```
class ActiveTest < Test::Unit::TestCase
  # Test para el nombre del activo
  def test_nombre_activo
```

```
    active1 = Active.create(:active => 'active001')
    assert_equal 'active001', active1.active
  end

  # Test para el url del activo
  def test_url_activo
    active1 = Active.create(:url => 'http://active1.deb')
    assert_equal 'http://active1.deb', active1.url
  end

  # Test para comprobar que no puede grabarse un activo sin parámetros
  def test_dont_should_save_active_withouth_params
    active1 = Active.new()
    assert !active1.save, "Guardado activo sin parametros"
  end

  # Test para comprobar que no puede guardarse un activo sin nombre
(validates_presence => true)
  def test_dont_should_save_active_withouth_name
    active1 = Active.new( #:active => 'active001',
      :url => 'http://active1.deb')
    assert !active1.save, "No se puede guardadar activo sin nombre"
  end

  # Test para comprobar que no puede guardarse un activo sin url
(validates_presence => true)
  def test_dont_should_save_active_withouth_url
```

```
    active1 = Active.new( :active => 'active001'
      #:url => "http://active1.deb"
    )
    assert !active1.save, "No se puede guardadar activo sin url"
  end

  #Test para comprobar que el nombre de un activo sea único
(validates_uniqueness => true)
  def test_active_is_not_valid_without_a_unique_name
    active1 = Active.new(:active => "active1",
                         :url => "http://active1.com")
    assert !active1.save
    assert_equal "has already been taken", active1.errors[:active].join('; ')
  end

  #Test para comprobar que el url de un activo sea único (validates_uniqueness
=> true)
  def test_active_is_not_valid_without_a_unique_url
    active1 = Active.new(:active => "active1",
                         :url => "http://active1.com")
    assert !active1.save
    assert_equal "has already been taken", active1.errors[:url].join('; ')
  end

end
```

Luego de escribir esta serie de pruebas unitarias, y realizadas al modelo “activo” del sistema se obtienen los siguientes resultados:

```
Loaded suite test/unit/active_test
Started
.......
Finished in 0.087913 seconds.
7 tests, 9 assertions, 0 failures, 0 errors
```

Lo que significa que de 7 pruebas realizadas, con 9 comparaciones, no falló ninguna; sin presentar ningún error en la escritura del código.

**Pruebas funcionales a los controladores.**

Siempre que se use un generador para crear un controlador con ActiveRecord, Rails automáticamente crea una prueba funcional para este en el directorio *test/functional* del proyecto. Las pruebas funcionales permiten verificar que nuestros controladores muestran el correcto comportamiento en las peticiones hechas por el usuario. Desde una prueba funcional el sistema puede también invocar el código de las vistas para comprobar las acciones correspondientes. A continuación se presentan las pruebas funcionales realizadas al sistema y sus resultados.

```
class ActivesControllerTest < ActionController::TestCase

  setup do
    @active1= Active.create(:active => 'active1', :url => 'http://active1.deb')
    @update= {:active => 'active2', :url => 'http://active2.deb'}
  end

  # Test para probar el ruteo de la vista index
  def test_should_get_index
    get :index
    assert_response :success
  end
```

```
  # Test para probar la acción nuevo activo
  def test_should_get_new_active
    get :new
    assert_response :success
  end

  # Test para probar el ruteo de la vista nube
  def test_should_get_nube
    get :nube
    assert_response :success
  end

  # Test para probar la acción crear activo
  def test_should_create_active
    post :create, :active1 => @update
    assert_response :success
  end

  # Test para probar la acción mostrar activo
  def test_should_show_active
    get :show, :id => @active1.to_param
    assert_response :success
  end

  # Test para probar la acción nuevo activo
  def test_should_get_edit_active
    get :edit, :id => @active1.to_param
```

```
    assert_response :success
  end

  # Test para probar la acción actualizar activo
  def test_should_update_active
    put :update, :id => @active1.to_param, :active1 => @update
    assert_redirected_to :action => "show"
  end

# Test para probar la acción eliminar activo
  def test_should_destroy_active
    delete :destroy, :id => @active1.to_param
    assert_redirected_to actives_path
  end
end
```

Luego de escribir esta serie de pruebas unitarias, realizadas a los controladores del sistema se obtienen los siguientes resultados:

```
Loaded suite test/functional/actives_controller_test
Started
.......

Finished in 0.70563 seconds.
8 tests, 8 assertions, 0 failures, 0 errors
```

Lo que significa que de 8 pruebas realizadas, con 8 comparaciones, no falló ninguna; sin presentar ningún error en la escritura del código.

### 3.4.2 Pruebas de aceptación.

Las pruebas de aceptación son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. Se debe especificar uno o diversos escenarios de prueba de aceptación para comprobar que una historia de usuario ha sido correctamente implementada [47].

Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de estas pruebas es garantizar que los requisitos han sido cumplidos y que el sistema es aceptable [36].

Las pruebas de aceptación son consideradas como “pruebas de caja negra” (“Black box system tests”). Asimismo, en caso de que fallen varias pruebas, se debe indicar el orden de prioridad de resolución. Una HU no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación [47]. A continuación se presentan las pruebas de aceptación realizadas a las diferentes HU y los resultados obtenidos para cada iteración.

**Casos de prueba de aceptación para la iteración 1.**

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU1_P1** | **Número de la HU:** 1 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Crear nuevo activo reutilizable de software. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de crear un nuevo activo reutilizable de software. | |
| **Condiciones de ejecución: -** | |
| **Entrada / Pasos de ejecución:** Se intenta adicionar un activo al sistema introduciendo datos válidos, no duplicados. Datos: nombre, url, etiquetas. | |
| **Resultado esperado:** Se crea un nuevo activo en la base de datos, con sus datos y etiquetas | |

| |
|---|
| correspondientes. Se muestra mensaje |
| **Evaluación de la prueba:** Prueba satisfactoria. |

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU1_P2** | **Número de la HU:** 1 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Modificar activo reutilizable de software. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de modificar un activo reutilizable de software. | |
| **Condiciones de ejecución:** El activo debe existir en la base de datos. | |
| **Entrada / Pasos de ejecución:** Se intenta modificar un determinado activo en el sistema cambiando, agregando o eliminando datos válidos, no duplicados. Datos: nombre, url, etiquetas. | |
| **Resultado esperado:** Se modifica el activo en la base de datos, con sus datos y etiquetas correspondientes. Se muestra mensaje. | |
| **Evaluación de la prueba:** Prueba satisfactoria. | |

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU1_P3** | **Número de la HU:** 1 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |

| |
|---|
| **Nombre de la prueba:** Eliminar activo reutilizable de software. |
| **Descripción de la prueba:** Prueba para la funcionalidad de eliminar un activo reutilizable de software. |
| **Condiciones de ejecución:** El activo debe existir en la base de datos. |
| **Entrada / Pasos de ejecución:** Se intenta eliminar un determinado activo en el sistema. El sistema pide confirmación. |
| **Resultado esperado:** Se elimina el activo en la base de datos, con sus datos y etiquetas correspondientes. Se muestra mensaje. |
| **Evaluación de la prueba:** Prueba satisfactoria. |

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU1_P4** | **Número de la HU:** 1 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Descargar activo reutilizable de software. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de descargar un activo reutilizable de software. | |
| **Condiciones de ejecución:** El activo debe existir en la base de datos. | |
| **Entrada / Pasos de ejecución:** Se intenta descargar un determinado activo en el sistema. | |
| **Resultado esperado:** Se descarga el activo en la ubicación deseada. | |
| **Evaluación de la prueba:** Prueba satisfactoria. | |

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU2_P1** | **Número de la HU:** 2 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Búsqueda de activo reutilizable de software. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de buscar un activo reutilizable de software mediante las etiquetas. | |
| **Condiciones de ejecución:** El activo debe existir en la base de datos y debe estar etiquetado. | |
| **Entrada / Pasos de ejecución:** Se seleccionan las etiquetas que se encuentran en la nube, la lista de etiquetas seleccionadas es mostrada, luego se procede a la acción de búsqueda. | |
| **Resultado esperado:** Los activos que concuerden con los criterios de búsquedas son mostrados. | |
| **Evaluación de la prueba:** Prueba satisfactoria. | |

**Casos de prueba de aceptación para la iteración 2.**

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU3_P1** | **Número de la HU:** 3 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Nube de etiquetas. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de generar la nube de etiquetas. | |
| **Condiciones de ejecución:** Los contextos y sus etiquetas correspondientes deben existir en la base | |

| |
|---|
| de datos. |
| **Entrada / Pasos de ejecución:** Se selecciona la opción de la nube de etiquetas, la lista de contextos y sus respectivas etiquetas es mostrada. |
| **Resultado esperado:** Al seleccionar una etiqueta determinada el sistema debe mostrar los activos pertenecientes a esta. |
| **Evaluación de la prueba:** Prueba satisfactoria. |

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU4_P1** | **Número de la HU:** 4 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Listar activos reutilizables de software. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de listar todos los activos. | |
| **Condiciones de ejecución:** Los activos deben existir en la base de datos. | |
| **Entrada / Pasos de ejecución:** Se selecciona la opción de listar activos. | |
| **Resultado esperado:** Todos los activos de la base de datos deben ser mostrados. | |
| **Evaluación de la prueba:** Prueba satisfactoria. | |

**Casos de prueba de aceptación para la iteración 3.**

| **Caso de prueba de aceptación.** | |
|---|---|
| **Código: HU5_P1** | **Número de la HU:** 5 |
| **Nombre de la persona que realiza la prueba:** Marcel Bauta. | |
| **Nombre de la prueba:** Buscar activos relacionados. | |
| **Descripción de la prueba:** Prueba para la funcionalidad de buscar todos los activos relacionados. | |
| **Condiciones de ejecución:** Los activos deben existir en la base de datos. | |
| **Entrada / Pasos de ejecución:** Se selecciona la opción de buscar activos relacionados. La lista con todos los activos es mostrada, se selecciona un activo. | |
| **Resultado esperado:** Todos los activos que existen en la base de datos, relacionados con el activo seleccionado deben ser mostrados. | |
| **Evaluación de la prueba:** Prueba satisfactoria. | |

**Conclusiones del capítulo 3.**

Durante la elaboración de este capítulo, la elaboración de las fases de Implementación y Pruebas propias de la metodología de desarrollo utilizada, permite arribar a las siguientes conclusiones:

- Durante la Fase de Implementación se describen las tareas de programación en correspondencia con cada una de las Historias de Usuarios identificadas.
- Se hace una descripción de los estándares de codificación empleados en el desarrollo del sistema permitiendo que futuras mejoras realizadas al sistema sean de fácil ejecución y la detección de errores una tarea menos engorrosa.

- Durante la Fase de Pruebas se realizan las pruebas unitarias y de aceptación garantizando que los requisitos han sido cumplidos y que el sistema es aceptable.
- Se generan los artefactos de los casos de pruebas de aceptación para tener constancia de que las HU fueron debidamente implementadas.

## CONCLUSIONES.

El cumplimiento de los objetivos trazados para el presente trabajo de diploma permite arribar a las conclusiones siguientes:

- La sistematización y el análisis integral de los elementos referidos a la gestión de búsquedas dentro de un repositorio de activos, las tecnologías y metodologías seleccionadas, así como de las concepciones teóricas que la sustentan, constituyen el punto de partida para ofrecer un buscador de activos de software para la gestión de búsqueda dentro del repositorio, que permita optimizar el proceso.
- La implementación del software mediador, con resultados satisfactorios, demostró su pertinencia y viabilidad, al optimizar el proceso de búsquedas, dentro de un repositorio de activos reutilizables de software.
- La aplicación de las pruebas unitarias, funcionales y de aceptación garantizó el cumplimiento de los requisitos, determinando que el sistema presenta una calidad aceptable y se generaron los artefactos de los casos de pruebas de aceptación.

Por todo lo anteriormente expresado se puede concluir que se cumplió con el objetivo general propuesto: Desarrollar un software mediador para la búsqueda de activos de software.

## RECOMENDACIONES.

Tomando como base la investigación realizada y los resultados obtenidos durante la realización del presente trabajo, se recomienda:

- Desplegar la solución en el CEDIN para que sus servicios puedan ser de utilidad en el trabajo con el repositorio de activos reutilizables de software.
- Agregarle el buscador funcionalidades semánticas.
- Integrar la solución como plugin del Redmine.

## BIBLIOGRAFÍA.

**Yao, H y Etzkorn, L**. *Towards a semantic-based approach for software reusable component classification and retrieval. Proceedings of the 42nd annual southeast regional conference : ACM Press.* págs. 110-115.

**Williams, J. Rails Solutions**, *Ruby on Rails Made Easy*. s.l. : Apress, 2007.

**Trigo, V.** Historia y evolución de los lenguajes de programación.

**Tercero, R**. *XP Programación Extrema para Desarrollo de Sistema Basados en Web*.

**Sommerville, I**. *Software Engineering*, 8th ed.

**Sánchez Rosado, R J, Hernández, J y Alférez, G H**. *Repositorio de Metadatos de Componentes de Software Reutilizables Para Desarrolladores de la Iglesia Adventista del Séptimo Día.*

**Ruby, S, Thomas, D y Heinemeier, D**. *Agile Web Development with Rails Fourth Edition.* s.l. : The Pragmatic Bookshelf, 2010.

**Rodríguez, N y Barbosa, L**. *On the Specification of a Component Repository*. 2004.

**Pressman, R.** *Ingenieria del Software, un enfoque práctico*. s.l. : Mc-Graw Hill, 2002.

**Perrotta, P.** *Metaprogramming Ruby The Pragmatic Bookshelf*. 2010.

**Palacio, J**. *Gestión de proyectos ágil: conceptos básicos*. 2006.

**Meyer, B**. *The significance of the components*. 1997.

**Mecella, M, y otros**. *A Repository of Workflow Components for Cooperative e-Applications*. 2003.

**Luján, S.** *Programación en Internet: Clientes Web (1ª edición).* s.l. : Editorial Club Universitario., 2001.

***Loeliger, J.*** *Version Control with Git*. s.l. : OREILLY, 2009.

**Lockhart, T**. *Tutorial de PostgreSQL*.

**Lidia, J M**. *Desarrollo de Software basado en componentes*. 2003.

**Lam Díaz, R M**. *Metodología para la confección de un proyecto de investigación*.

**Kruchten, P**. *The Rational Unified Process: An Introduction.* s.l. : Addison Wesley, 2000.

**Kindelan, R**. *Pequeño Tutorial de Lenguaje Ruby*. UCI : s.n., 2009.

**Jacobson, I, Booch, G y Rumbaugh, J**. *El Proceso Unificado de Desarrollo de Software*. s.l. : Addison Wesley, 2000.

**Holzner, S.** *Beginning Ruby on Rails*. s.l. : Wiley, 2007.

**Hernández, R A y Coello, S.** *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA*. La Habana : Editorial Universitaria, 2011.

**Guillén, D F**. *Ruby Fácil*. 2007.

**García, E, y otros**. *Una propuesta para la reutilización de componentes en el proceso de desarrollo de software educativo.*

**Fowler, M y Sccott, K**. *UML Gota a Gota*. 1999.

**Fowler, Ch***. Rails Recipes*. s.l. : The Pragmatic Bookshelf, 2006.

**Fernández, F A**. *Consideraciones sobre el desarrollo basado en componentes en la Red Cubana de Ciencia*. 2008. pág. 20.

**Fernandez, O**. *THE RAILS WAY*. s.l. : Addison-Wesley, 2008.

**Crespo, Y, Laguna, M A y Pérez, F J.** *Integrando un modelo de reutilización en la producción de software: entorno distribuido para el desarrollo basado en reutilización*. 2004.

**Clark, M***. Advanced Rails Recipes*. s.l. : The Pragmatic Bookshelf, 2008.

**Benson, E**. *The Art of Rails*. s.l. : Wiley, 2008.

**Alameda, E**. *Practical Rails Projects*. s.l. : Apress, 2007.

**Tutorial del PostgreSQL (Breve historia de Postgres).** [En línea] *http://postgresql.uci.cu/foro/viewtopic.php?f=6&t=273.*

**Sitio oficial en español de Mozilla Firefox**. [En línea] http://www.mozilla-europe.org/es/products/firefox/.

**Presente y Futuro de la Reutilización de Software**. [En línea] *http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=reuse.*

**Aptana, IDE para aplicaciones Ajax.** [En línea] *http://www.genbeta.com/web/aptana-ide-para-aplicaciones-ajax .*

## REFERENCIAS BIBLIOGRÁFICAS.

1. **Ajila, S A.** *Change Management: Modeling Software Product Lines Evolution, Department of Systems & Computer Engineering, Carleton University.* 2005.

2. **Montilva, J A.** *Desarrollo de Software Basado en Líneas de Productos de Software, Universidad de Los Andes. Facultad de Ingeniería. Departamento de Computación* . Mérida – Venezuela : IEEE Computer Society, 2006.

3. **Krueger, C W.** *New methods in Software Product Line practice. Examining the benefits of next-generation SPL methods.* 2006. pág. 40.

4. **Groher, I, Krueger, C W y Schwanninger, C.** *A Tool-Based Approach to Managing Crosscutting Feature Implementations.* s.l. : AOSD, 2008.

5. **Krueger, C W.** *Towards a Taxonomy for Software Product Lines.* 2003. pág. 11.

6. SourceForge. Find, Create, and Publish Open Source software for free. [En línea] http://sourceforge.net/about.

7. The Nuts & Bolts of Project Kenai. [En línea] http://kenai.com/projects/help/pages/KenaiOverview.

8. **Collins-Sussman, B, Fitzpatrick, W y Pilato, M.** *Control de versiones con Subversion.*

9. **Nagel, W.** *Using The Subversion Version Control System in Development Projects.*

10. **Méndez, N; Elvia, M;.** *Modelo de evaluación de metodologías para el desarrollo de software.* Caracas : s.n., 2006.

11. **Canal, C.** *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software.* Universidad Málaga : s.n., 2000.

12. **Szypersky, C.** *Component Software. Beyond Object-Orientend Programming.* 1998.

13. **Anderson, W, y otros.** *COTS and Reusable Software Management Planning: A Template for Life-Cycle Management.* 2007.

14. **Rojas, M A y García, J C M.** *Introducción y principios básicos del desarrollo de Software basado en componentes.* 2004. pág. 12.

15. **Montilva, J A.** *Desarrollo de Software Basado en Componentes.* Santa Cruz, Bolivia : s.n., 2003. pág. 36.

16. **Sommerville, I.** *Software engineering.* s.l. : Addison-Wesley, 2000. 6ta edición.

17. **Sametinger, J.** *Software engineering with reusable components.* s.l. : Springer Verlag, 1997.

18. **Sodhi, J y Sodhi, P.** *Software reuse: Domain analysis and design process.* s.l. : McGraw-Hill, 1999.

19. **Gaedke, M, Rehse, J y Graef, G.** *A Repository to facilitate Reuse in Component- Based Web Engineering.* 1999.

20. **Pérez, I.** *Propuesta de metodología para el diseño e implantación de repositorios de activos de software reutilizables.* 2011.

21. **Yunwen , Y y Fischer, G.** *Context-Aware Browsing of Large Component Repositories.* 2001. pág. 8.

22. **Matsumoto, Y.** *A software factory: An overall approach for software production.* s.l. : P Freeman, 1990.

23. **Prieto-Diaz, R.** *Implementing faceted classication for software reuse.* 1991.

24. **Ostertag, E, y otros.** *Computing similarity in a reuse library system. ACM Transaction on Software Engineering.* 1992.

25. **Henninger, S.** *Using iterative refinement to find reusable software.* 1994.

26. **Figueroa, R G, Solís, C J y Cabrera, A A.** *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* Universidad Técnica Particular de Loja : s.n.

27. **Palacio, J.** *El modelo Scrum.* 2006.

28. **Rising, L y Janoff, N S.** *The Scrum Software Development Process for Small Teams.* 2000.

29. **Letelier, P y Penadés, M C.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)* . Universidad Politécnica de Valencia : s.n.

30. **Larrman, C.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : Prentice Hall, 1999.

31. **Pressman, R.** *Ingeniería del Software, un enfoque práctico.*

32. Herramientas de programación. [En línea] 2009. http://www.lenguajes-de-programacion.com/herramientas-de-programacion.shtml.

33. **Cerda, F.** NetBeans 6.5 El único IDE que necesitas. [En línea] 2009. http://www.slideshare.net/felipecerda/netbeans-el-nico-ide-que-necesitas.

34. World Wide Web Consortium (W3C). [En línea] http://www.w3c.es/.

35. **Catalán García-Manso, A M.** *Análisis, diseño e implementación de un sitio Web Departamental. Creación, modificación y almacenamiento de contenidos.* 2009.

36. **BECK, K.** *Planeando en Programación Extrema.* 2000.

37. **Valdés, D P.** Los diferentes lenguajes de programación para la Web. [En línea] 2007. http://www.maestrosdelWeb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-Web/.

38. **Jeffries, R, Anderson, A y Hendrickson, C.** *Extreme Programming Installed.* s.l. : Addison-Wesley, 2002.

39. **Wake, W C.** *Extreme Programming Explored.* s.l. : Addison-Wesley, 2002.

40. **Newkirk, J y Martin, R C.** *Extreme Programming in Practice.* s.l. : Addison-Wesley, 2001.

41. **Beck, K.** *Extreme Programming Explained. Embrace Change.* s.l. : Pearson Education Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

42. **Bahit , E.** *POO y MVC en PHP. El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC.*

43. **Pantoja, E.** *El patrón de diseño Modelo-Vista-Controlador (MVC).*

44. **Lago, R.** *Patrones de diseño software.* 2007.

45. **Gamma, E, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1995.

46. **Allende, R.** *Desarrollo de Portales y Extranet con Plone.* 2006.

47. **Joskowicz, J.** *Reglas y Prácticas en eXtreme Programming.* 2008.

48. FUNCTIONAL TESTING - PRUEBAS FUNCIONALES . [En línea] http://www.calidadysoftware.com/testing/pruebas_funcionales.php.

## GLOSARIO DE TÉRMINOS.

**TIC(s)**: Tecnologías de la información y la comunicación, agrupan los elementos y las técnicas usados en el tratamiento y la transmisión de la información, principalmente la informática, Internet y las telecomunicaciones.

**CEDIN**: Centro de Informática Industrial, con el objetivo de desarrollar productos y servicios informáticos de automatización industrial y computación gráfica, con un alto valor agregado y que cumplan las necesidades y expectativas de los clientes, potenciando la formación especializada y la investigación.

**Synaptic**: Programa informático que proporciona una interfaz gráfica GTK+ a APT, para el sistema de gestión de paquetes de Debian GNU/Linux. Generalmente se utiliza para sistemas basados en paquetes *.deb.

**Redmine**: Herramienta para la gestión de proyectos que incluye un sistema de seguimiento de incidentes con seguimiento de errores. Otras herramientas que incluye son calendario de actividades, diagramas de Gantt para la representación visual de la línea del tiempo de los proyectos, wiki, foro, visor del repositorio de control de versiones, RSS, control de flujo de trabajo basado en roles, integración con correo electrónico, etcétera.

**Framework**: Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes.

**JRuby**: Implementación 100% Java del lenguaje de programación Ruby.

**GPL**: Licencia Pública General de GNU/Linux, es una licencia creada por la Free Software Foundation en 1989 (la primera versión, escrita por Richard Stallman), y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

**Basecamp**: Programa de gestión de proyectos vía web. Su espacio de trabajo ofrece listas de tareas, un editor de documentos colaborativo, hitos, archivos compartidos, registro de tiempos y mensajería interna, organizador y gestor de tareas.

**Licencia BSD**: Licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Al contrario que la GPL permite el uso del código fuente en software no libre.

**Rational Software**: Compañía fundada por Paul Levy y Mike Devlin en 1981 con el nombre de Rational Machines, para proporcionar herramienta que expandieran las prácticas modernas de ingeniería de software, particularmente la arquitectura modular y el desarrollo iterativo. Actualmente conocida como una familia de software de IBM para el despliegue, diseño, construcción, pruebas y administración de proyectos en el proceso desarrollo de software.

**IBM**: International Business Machines es una empresa multinacional estadounidense de tecnología y consultoría. Fabrica y comercializa hardware y software para computadoras, y ofrece servicios de infraestructura, alojamiento de Internet, y consultoría en una amplia gama de áreas relacionadas con la informática, desde computadoras centrales hasta nanotecnología.

**HTTP**: Hypertext Transfer Protocol, protocolo usado en cada transacción de la web. Está orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

**OSI**: Open System Interconnection, modelo de red descriptivo creado por la Organización Internacional para la Estandarización en el año 1984. Es decir, es un marco de referencia para la definición de arquitecturas de interconexión de sistemas de comunicaciones.

## ANEXOS

Anexo # 1 Diagrama de clases del sistema.

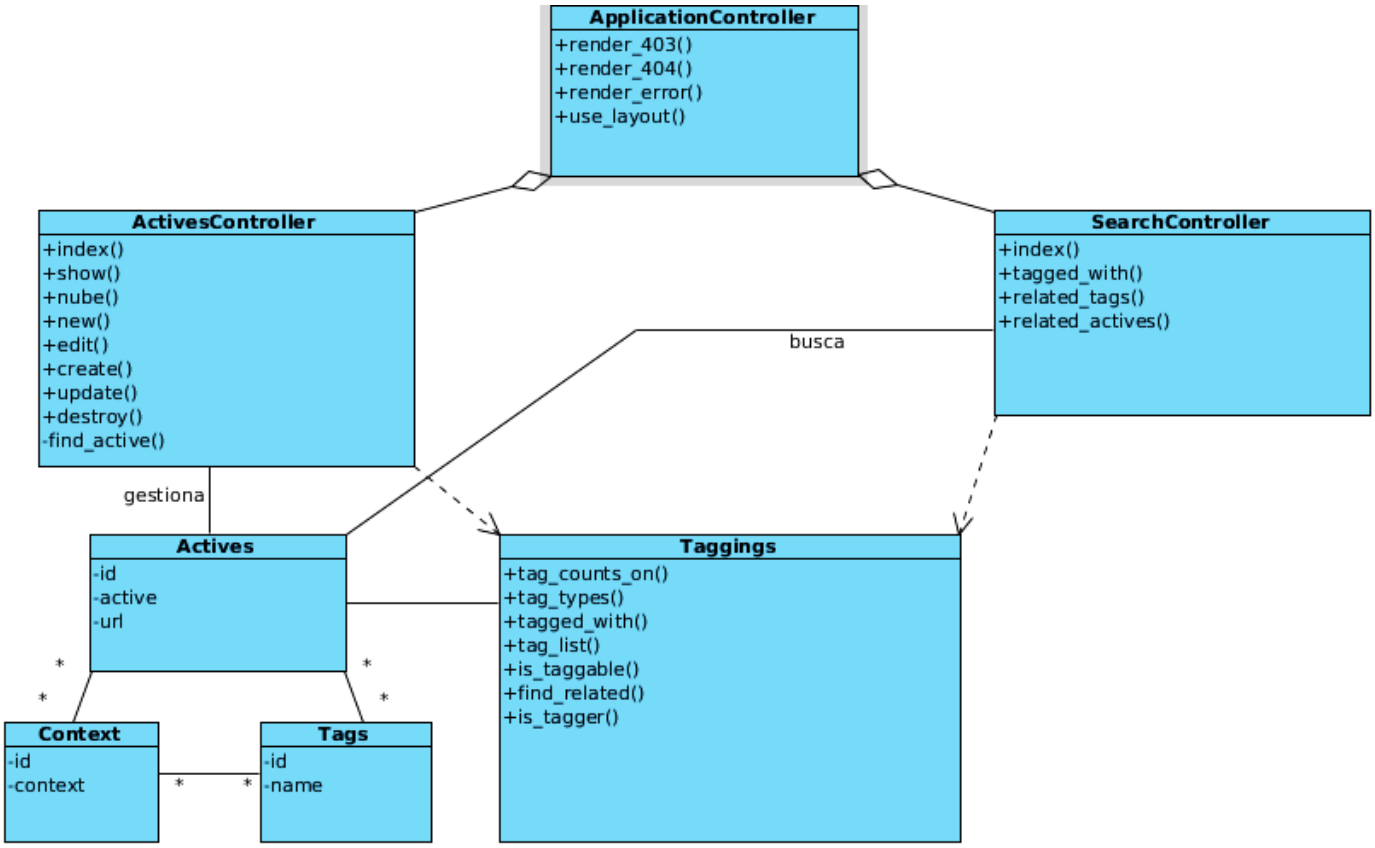



***Ilustración 7 Diagrama de clases del sistema.***

Anexo # 2 Diagrama de despliegue.

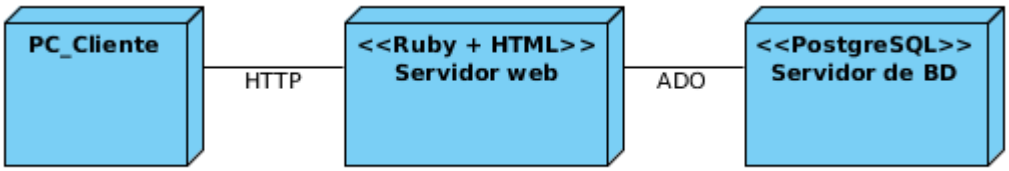



***Ilustración 8 Diagrama de despliegue.***