# Universidad de las Ciencias Informáticas


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

## Título: Gitadmin. Herramienta para la administración de los repositorios Git.

**Autor:** Iván Martínez Pupo

**Tutores:** Ing. Anielkis Herrera González.
Lic. Graciela González Pérez.

-- de Junio 2010.
Año del 51 Aniversario del Triunfo de la Revolución

# Declaración de autoría

Declaración de Autoría _____

Declaro ser el único 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 ____ del año_____.

_____
Firma del Autor:
Iván Martínez Pupo

_____ _____

Firma del Tutor:
Anielkis Herrera González

Firma del Tutor:
Graciela González Pérez

# Agradecimientos

Agradezco a mi tutora Graciela por el empeño y la confianza.
A mi tía Olga por el esfuerzo.
A mi mamá y a mi papá por la perseverancia.
A la revolución por crear esta universidad maravillosa.

# Dedicatoria

A mi tía Eloísa que aunque este lejos es como si estuviera cerca.

A mi mama por nunca abandonar y ser perseverante aun en las causas perdidas.

A mi papa por enseñarme a pensar.

A mi hermano porque tengo confianza que será mejor que yo en todo lo que se proponga.

A mi novia linda Hazzel por estar siempre a mi lado.

A mi familia en general por apoyarme y ser tan comprensiva.

A mis amigos, no hace falta especificar ellos saben quienes son.

**Resumen**

El proyecto Nova de la Universidad de las Ciencias Informáticas (UCI) se encuentra en estos momentos desarrollando el sistema operativo basado en GNU/Linux y el cual cuenta con múltiples líneas de producción. La actual cantidad de líneas de trabajo ha propiciado un aumento de repositorios Git[1], sin embargo en el proyecto no se cuenta con una herramienta para la administración de los mismos.

En este trabajo se presenta una herramienta como propuesta para administrar los repositorios Git del proyecto Nova. Con esta herramienta se pretende solucionar la necesidad existente de administrar los repositorios Git, usuarios y permisos con restricción de acceso a nivel de ramas mediante una interfaz amigable e intuitiva.

Palabras Claves: Repositorio, Gestor de repositorios.

[1] Es un Sistema de Control de Versiones desarrollado por Linus Torvalds, creado inicialmente para mantener el kernel de GNU/Linux.

**Tabla de Contenido**

## Introducción

Desde finales del siglo XX las sociedades contemporáneas se han visto involucradas en grandes transformaciones debido al vertiginoso desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC). Su uso impulsa todos los campos del conocimiento y provoca una revolución tecnológica que influye en los más disímiles aspectos de la vida cotidiana, la cultura, la economía y los procesos sociales en general.

Cuba, independientemente de no encontrarse entre los países desarrollados incorpora el uso de las TIC a la vida cotidiana del pueblo. Se propone llevar a la sociedad cubana la revolución informática, haciendo uso de los principales medios de comunicación y de la computación para llevar adelante el conocimiento, creando los Joven Clubs de Computación y llevando a las escuelas las computadoras para realizar los procesos de aprendizaje mucho más interactivos. Entre otros objetivos se trazó como meta alcanzar un nivel relevante en la producción de software a nivel internacional.

Con el objetivo de llevar adelante la producción de software en Cuba surgieron diferentes alternativas, una de las más importantes fue la creación de la Universidad de las Ciencias Informáticas (UCI). La misma fue creada en el año 2003 y en la actualidad consta de 9 facultades, 3 facultades regionales y estudiantes provenientes de todos los municipios del país. Entre sus principales objetivos se encuentran llevar adelante la revolución informática en la sociedad cubana y la producción de software de calidad.

La producción de software de calidad se convierte en una tarea compleja debido al bloqueo impuesto por los E.U y a las constantes agresiones a las que se ha visto sometida Cuba. Esta hostilidad ha obligado a la isla a proponerse alcanzar la independencia tecnológica migrando hacia el software libre, permitiéndose de esta manera desarrollar herramientas orientadas a la satisfacción de las necesidades sociales y adaptadas a las condiciones reales de la sociedad cubana.

Con el objetivo de alcanzar la independencia tecnológica en Cuba surgió el proyecto Nova en la UCI. Nova fue creado en octubre del 2004 con el objetivo de crear un sistema operativo cubano personalizable que respondiera a las necesidades reales del país. Para Cuba es un proyecto importante debido a que se encuentra a la cabeza de la migración hacia el software libre y constituye un camino a la independencia tecnológica.

El desarrollo de software en Nova se implementa organizado en grupos de trabajo. Este trabajo conjunto requiere de coordinación, uso de herramientas y aplicaciones comunes para que el software desarrollado por elementos o módulos pueda trabajar de forma sincronizada. En un proyecto tan abarcador como Nova y que se encuentra en constante crecimiento, se hace necesario ayudar al proceso de producción de software con

herramientas que permitan establecer un control y diferentes versiones para un mismo software o código fuente.

Las herramientas de control de versiones sirven para mantener distintas versiones de código fuente, documentación, imágenes o ficheros de configuración entre otras. Permite el trabajo simultáneo de desarrolladores sobre un fichero y conocer quién está modificando el software, y dónde. También realiza un registro histórico que permite regresar a una versión anterior en caso de necesidad. [2]

Las herramientas para el control de versiones juegan un rol importante en la producción de software en el proyecto Nova. Éstas son las responsables de que se pueda desarrollar software concurrentemente y mantener un historial del proyecto. En la actualidad el proyecto Nova utiliza Git como herramienta de control de versiones. Esta herramienta presentó su primer prototipo el 7 de abril del 2005, fue creada por Linus Torvalds con excelentes prestaciones, y se ha convertido en uno de los Sistemas de Control de Versiones (SCV) más populares entre los desarrolladores y auditores de software.

Partiendo del estado actual del proyecto Nova se ha identificado una **situación problemática** que es la que sirve de partida para el desarrollo de esta investigación. El proyecto Nova desde su inicio ha ido creciendo en importancia, en magnitud como sistema operativo y en tamaño como proyecto. Este crecimiento como sistema operativo y como proyecto productivo ha traído consigo el aumento en número de repositorios Git, usuarios y permisos. Sin embargo, el proyecto no cuenta con una herramienta para administrar cantidades crecientes de repositorios Git, usuarios y permisos, y que además posea un sistema de administración a nivel de ramas.

Los actuales métodos de administración se realizan en un terminal y directamente sobre los ficheros de configuración. Por tanto, es una necesidad la creación de una herramienta de interfaz visual que posibilite la administración de los repositorios Git con gran cantidad de usuarios, repositorios y permisos de forma eficiente, lo que justifica la necesidad de la presente investigación.

La anterior situación problemática conduce al siguiente **problema científico**:

¿Cómo administrar de forma eficiente grandes cantidades de usuarios, repositorios y permisos sin necesidad de trabajar directamente en un terminal y sobre los ficheros de configuración?

Partiendo del problema anterior se establece como **objeto de estudio** los Sistemas Gestores de Repositorios de control de fuentes y revisiones de tipo Git, y como **campo de acción** las aplicaciones con interfaz gráfica para administrar repositorios de código fuente y revisiones de tipo Git.

Se define como **objetivo general** desarrollar una herramienta de interfaz visual que administre de forma sencilla grandes cantidades de usuarios, repositorios y permisos en los

repositorios Git.

Se plantearon los siguientes **objetivos específicos**:

- Sistematizar los antecedentes y el estado actual de los Sistemas Gestores de Repositorios Git.
- Sistematizar las herramientas, tecnologías y la metodología de desarrollo de software que pudiera ser de utilidad para la creación de la herramienta.
- Modelar e implementar una interfaz visual que permita la administración de repositorios Git.

Para dar cumplimiento al objetivo general y los objetivos específicos se plantearon las siguientes **tareas de investigación**:

- Sistematización de antecedentes y el estado actual de los Sistemas Gestores de Repositorios Git.
- Sistematización de las herramientas, tecnologías y la metodología de desarrollo de software que pudiera ser de utilidad para la creación de la herramienta.
- Modelación del diseño de la herramienta propuesta.

Partiendo de lo planteado anteriormente se establece como la **idea a defender** la siguiente: una herramienta de interfaz gráfica para administrar grandes cantidades usuarios, repositorios y permisos de los repositorios Git, propiciaría una administración más eficiente y simple de los repositorios Git para la producción de software en Nova.

Para llevar a cabo esta investigación se utilizan los siguientes **métodos científicos**:

Método Bibliográfico: Mediante este método se hizo un estudio de una variada documentación referente a los gestores de repositorios Git para obtener de ella análisis, experiencias y sugerencias que pudieran ser incorporadas a la tesis. Con el uso de este método se estudió además la documentación referente a los potenciales lenguajes de programación para desarrollar la herramienta, así como las experiencias de otros desarrolladores en el campo de acción que se desarrolla la herramienta.

Histórico-Lógico: Permitió obtener una mayor comprensión del estado actual de las aplicaciones de interfaz visual para los gestores de repositorios Git, los lenguajes en los cuales se ha desarrollado, su evolución, el uso práctico de los gestores de repositorios Git y la tendencia de su desarrollo en el tiempo.

Modelación: Este método permitió realizar una representación de la situación que se analiza. Permitió obtener mediante diagramas y objetos una mayor compresión del problema y desarrollar un modelo para la aplicación a desarrollar a partir de la situación problemática. La modelación permitió llegar a obtener una propuesta de solución que pudiera responder a las necesidades del proyecto Nova.

Con la implementación del producto final se logrará:

- Una interfaz grafica intuitiva para administrar los repositorios Git.
- Una API para el manejo de repositorios Git reutilizable para otros gestores de repositorios Git como Redmine.
- Ventajas de administración remota mediante acceso web.
- Mayor agilidad en el proceso de administración de repositorios Git en Nova.

La presente investigación consta de una introducción, tres capítulos, conclusiones generales, recomendaciones, referencias bibliográficas, bibliografías y anexos.

**Capítulo 1:** Sistemas Gestores de Control de Versiones Git.

En este capítulo se desarrolla la investigación sobre los sistemas de control de versiones haciendo énfasis en los SCV de tipo Git. Se hace un realiza un estudio de los antecedentes de los sistemas gestores de repositorios Git. Se analizan las herramientas con interfaces visuales para gestores de repositorios Git que facilitan la administración de múltiples usuarios, repositorios y permisos.

**Capítulo 2:** Entorno de Desarrollo.

Este capítulo está compuesto por el análisis de los principales elementos del entorno de desarrollo. Se definen y describen aquellas herramientas y tecnologías libres que auxiliarán el proceso de construcción de la aplicación. Se establecen mediante análisis los lenguajes que se utilizaran tanto para la programación como para el modelado de la solución propuesta. Se determina la metodología de desarrollo de software que asistirá el proceso de creación de la herramienta informática acorde a las necesidades de la presente investigación.

**Capitulo 3:** Descripción y Desarrollo de la Solución Propuesta.

En este capítulo se realizará el desarrollo ágil de Aplicación Informática presentada como solución. Se muestran las primeras fases del desarrollo de la aplicación informática, así como los artefactos generados considerados de importancia para obtener un correcto entendimiento del diseño de la aplicación. Además se describen los casos de pruebas o pruebas de aceptación realizados para cada historia de usuario en cada iteración, lo que permite ir liberando en cada iteración pequeños componentes funcionales de la aplicación.

Capítulo

# Sistemas Gestores de Control de Versiones Git.

El uso de un sistema de control de versiones (SCV) en Nova en estos momentos se encuentra limitado, debido a la inexistencia de una herramienta para administrar los repositorios Git. En este capítulo se desarrolla el estudio y análisis de los antecedentes de los SCV Git, así como de algunos gestores de repositorios Git que pudieran ser de utilidad para resolver el actual problema de administración en Nova. Se pretende determinar aquellas cualidades que pudieran ser de utilidad para la construcción de un sistema que administre múltiples sistemas de de control de versiones Git, usuarios y permisos de Nova.

## Introducción a los Sistemas de Control de Versiones

Un SCV es un sistema para mantener un historial de versiones anteriores de un contenido determinado, comúnmente código fuente, ficheros de configuración o documentación. Estos sistemas permiten guardar diversas copias de un fichero, crear scripts para automatizar copias, controlar fechas, usuarios de configuración y los datos de los que realizan copias en una línea de versiones. Los desarrolladores también pueden deshacer los cambios hechos en un momento dado o recuperar versiones pasadas, ver los datos históricos de cambios y comentarios de las versiones realizadas por otros desarrolladores y pueden también comparar diferentes versiones de archivos.

Uno de los objetivos más importantes de este tipo de herramienta es la colaboración, permitiendo a diversos desarrolladores trabajar sobre un mismo proyecto. Son ejemplos de sistemas de control de versiones: CVS, Subversion, BitKeeper, Perforce, Mercurial y Git. La presente investigación se encuentra particularmente orientada al uso de repositorios Git.

Los repositorios de código fuente se pueden dividir según su forma de trabajo con repositorios remotos. Estos quedan divididos de la siguiente forma:

- **Centralizado**

En un SCV centralizado todas las fuentes y sus versiones se almacenan en un único directorio (repositorio de fuentes) de un servidor (servidor central). Los integrantes del equipo de desarrollo que realizaran trabajos con esas fuentes, deben solicitar al SCV una copia local para trabajar. En esta copia realizan cambios y cuando estos cambios estén listos se le informa al SCV para que proceda a guardar los fuentes modificados como la última versión.

Cada copia de trabajo se comunica con el repositorio central tanto para leer como para almacenar en él. El inconveniente del uso de esta herramienta es que se necesita tener siempre una conexión al servidor o acceso a redes, esto permite guardar los cambios cada vez que se trabaja, se hace una revisión, se lee o almacena. Ejemplos: CVS, Subversion.

- **Distribuido**

Los sistemas de control de versiones distribuidas no cuentan con un repositorio central. Todos los integrantes de los equipos de trabajo tienen en su poder una copia propia del repositorio, todas las versiones y el historial. Los SCV distribuidos permiten la sincronización entre desarrolladores de sus repositorios. Si una de las versiones ha sido modificada en determinados fuentes y otra versión no, entonces las fuentes modificadas se convierten la versión más moderna. Cuando dos versiones coinciden con las mismas fuentes modificadas, el SCV pone en alerta a los desarrolladores para que decidan los cambios válidos.

En este SCV, cada repositorio local o desarrollador funciona como un nodo y donde cada nodo puede sincronizar con otros nodos. En este tipo de SCV todos pueden leer de la rama principal, aunque no todos puedan escribir en ella. Los desarrolladores pueden intercambiar cambios y actualizar unos de otros. Este sistema puede tener varias líneas en paralelo, aunque esto pudiera ser un inconveniente porque nunca existe una versión final.

Ejemplos: Mercurial, Bazaar, Arch, Git.

**Sistema de Control de Versiones Git.**

En estos momentos, el proyecto Nova de la Universidad de las Ciencias Informáticas tiene sus repositorios sobre un sistema de control de versiones de tipo Git. Este es un SCV que ha ido alcanzando gran popularidad a nivel mundial y cuenta con una característica que lo hace destacar de casi cualquier otro SCV y es su modelo de ramas. La gran mayoría de los SCV recomiendan que la mejor rama sea una copia del repositorio en un nuevo directorio. Git, por el contrario, permitirá tener múltiples ramas locales que son independientes entre sí. Git permite crear ramas de prueba, volver a un punto de bifurcación de una versión, aplicar

un parche y regresar a la rama de prueba y fusionar ambas. Git implementa un sistema que permite tener una rama que contiene todo el trabajo de producción, otra rama con el trabajo para hacer pruebas (testear) y ramas más pequeñas para otros trabajos secundarios. Git tiene la posibilidad de hacer entregas parciales a repositorios remotos, dando la posibilidad de subir solo las ramas compartidas.

Git permite el trabajo sin necesidad de estar conectado al repositorio central, esta es unas características con que cuentan la mayoría de los SCV distribuidos pero a diferencia de estos, esta herramienta cuenta con una gama mucho mayor de comandos a utilizar sin necesidad de una sincronización con la rama principal del servidor remoto. Con Git se hace una descarga de toda la información del servidor antes de terminar la conexión y hacer comparaciones, uniones de código y ver el historial de todos los datos que están en el servidor aunque no forme parte de las ramas locales.

Git, en comparación con otros SCVs como Subversion y Perforce es mucho más rápido; esto se debe a que la mayoría de las operaciones se hacen localmente. Otra de las principales razones es que la herramienta está escrita en el lenguaje C y fue construida para trabajar en el kernel de Linux; razón por la cual desde el principio ha sido capaz de mover de manera efectiva repositorios de gran tamaño. En comparación con otros SCVs como Subversion y CVS ocupa menos espacio.

**Sistemas de administración para repositorios de SCV**

Usualmente, cuando se trabaja sobre un solo proyecto es muy fácil hacer uso de los SCV, pero cuando una empresa necesita desarrollar muchos proyectos con muchos desarrolladores, la administración de los mismos se complica. Por tal motivo, se crearon herramientas para administrar múltiples repositorios de SCV.

Para realizar una correcta administración de los sistemas de control de versiones se debe contar con una serie de funcionalidades que permita controlar quienes acceden, a donde acceden y que modifican, así como el permiso de acceso para leer, escribir o modificar los diferentes ficheros que se encuentran en el repositorio.

Estos sistemas gestores deben permitir la gestión de los grupos de usuarios y repositorios. Dentro del proceso de gestión de grupos de usuarios y repositorios se encuentran los procesos de adicionar, eliminar y modificar los grupos, tanto de usuarios como de repositorios. Deben ser capaces de gestionar los permisos, determinando el tipo de acceso, si es de lectura o escritura, a que repositorio o grupo de repositorio se tiene acceso y que grupo de usuario o usuario en particular tiene a acceso a los mismos.

En el caso particular de la aplicación que se necesita en el proyecto Nova, es necesario que

permita una administración remota a los repositorios. Que facilite mediante una interfaz gráfica intuitiva la administración de múltiples usuarios, repositorios y permisos, permitiendo definir el acceso a ramas y optimizar de esta forma el proceso de administración de los repositorios de tipo Git.

**Herramientas para la gestión de repositorios Git.**

A continuación se realiza la investigación y análisis de las principales características de algunas herramientas que se encuentran involucradas en la gestión de repositorios Git.

**Kenai**

Kenai es un gestor de repositorios creado con JRuby y sistema de bases de datos MySQL con soporte para Mercurial, Subversion y Git. Esta plataforma liberada el 17 de septiembre del 2008 ofrece su sistema de administración a proyectos de fuente abierta. Posee área de descarga para documentos del proyecto, chat integrado para la colaboración directa de los implementadores, foros de discusión y listas de correo para seguimiento de hilos de conversación de foros y errores. Kenai ofrece un sistema para búsquedas de desarrolladores y proyectos mediante nube de etiquetas, permite el etiquetado de proyectos, noticias y errores. Personalización de páginas de contacto y de seguimiento de proyectos individuales [10].

Este posee integración con NetBeans IDE versión 6.7 permitiendo navegar por el proyecto en desarrollo y hacer seguimiento de versiones, así como aplicar parches nuevos y ver que otros desarrolladores se encuentran trabajando en línea. Esta plataforma tiene como desventaja ser relativamente nueva, por tal motivo cuenta con muy poca documentación e insuficiente experiencias de uso. Es una herramienta orientada totalmente a la revisión de códigos y al desarrollo de los mismos.

**Indenfero**

Esta es una aplicación para la administración de repositorio registrada a nombre de Ceondo Ltd, creada en el verano del 2008 e implementada en php. Entre sus principales características:

- Sistema de seguimiento de errores.
- Documentación colaborativa.
- Área de descargas.
- Buscador de código para Git, Subversion y Mercurial.
- Sistema de revisión de código.

- Sistema de administración avanzada para la administración personalizada de proyectos.
- Sistema de búsquedas.
- Sistema de visualización para cambios en el código entre versiones.
- Sistema de acceso de control para proyectos públicos y privados.
- Soporte multilenguaje.
- Sistema de subrayado para archivos de texto.
- Sistema de seguimiento por correo de cambios y errores.
- Personalización de línea de tiempo para actividades.
- Sistema de etiquetado para descargas, proyectos y documentación.

Esta herramienta cuenta con la desventaja de poseer poca documentación ya que es relativamente nueva. No cuenta con un sistema de administración orientado a la administración de repositorios y permisos a nivel de ramas [11].

**Redmine**

Redmine es una aplicación web desarrollada sobre Ruby on Rails con licencia General Public License (GPL) para la administración flexible de repositorios Git. Esta aplicación permite a los desarrolladores involucrados en los proyectos definir los hitos del proyecto y las tareas para cada uno de los hitos, así como un sistema para la asignación de tareas a los desarrolladores. A continuación se muestran algunas de sus principales características:

- Soporte para múltiples proyectos.
- Control de acceso basado en roles.
- Sistema flexible para el seguimiento de errores.
- Contiene diagrama de Gantt y calendario.
- Sistema de administración de archivos, documentos y noticias.
- Sistema de notificación email.
- Sistema para la gestión de wikis y foros de discusión.
- Trazado de línea.
- Campos personalizados para proyectos, usuarios y seguimiento de errores.
- Sistema de integración con SVN, CVS, Git, Mercurial, Bazaar and Darcs.
- Creación de email personalizado por proyecto.
- Soportes para autentificación de múltiples directorios activos.
- Soporte multilenguaje.
- Soporte para mantenimiento de múltiples bases de datos.

Entre sus principales desventajas se destaca que es un gestor de repositorios orientado

totalmente al desarrollo colaborativo, no está orientado a la administración de los repositorios. Su principal objetivo es garantizar el control sobre las versiones, llevando un control sobre los cambios en el código, cuando y porque desarrollador ha sido modificada una versión [12].

**Gitorious**

Gitorious es una herramienta escrita sobre Ruby en el framework Ruby on Rails desarrollada para entornos web y fue creada por Johan Sorense a finales del año 2007. Este gestor de repositorios Git fue creado para almacenar y administrar proyectos privados y públicos con fuente abierta. La entidad central de Gitorious es un proyecto que contiene uno o más repositorios Git administrados por los contribuyentes de los proyectos.
Este ofrece la posibilidad a los administradores de los proyectos inscritos de observar y controlar los cambios que tienen los proyectos que llevan, estos cambios incluyen nuevas versiones, cambios en el código, actualización y reparación de errores por parte de los contribuyentes. Características:

- Almacena y administra repositorios.
- Administración de clones de proyecto.
- Proyectos para creación de wikis.
- Maneja propuestas para combinar proyectos y revisión de proyectos.
- Realiza una línea de tiempo para actividades.
- Mantiene perfiles personalizados y línea de tiempo para actividades personales.
- Sistema de notificación.
- Libre para proyecto de fuente abierta.
- Soporte comercial y personalizado.

Este proyecto permite la fácil colaboración y comunicación entre los desarrolladores, facilitando la notificación de los cambios desarrollados a los involucrados de un proyecto. Entre sus desventajas se encuentra que al igual que Redmine no es un gestor de repositorios orientado a la administración, se enfoca directamente sobre la producción y el desarrollo de los proyectos que contiene [13].

**Gitosis**

Gitosis fue creado por Tommi Virtanen,esta herramienta tiene como objetivo realizar una administración de los repositorios de código de tipo Git de forma fácil y eficiente. Gitosis administra múltiples repositorios bajo una cuenta única y usa un sistema de llaves públicas y privadas para la autentificación de usuarios. Esta herramienta restringe el uso de algunos comandos para los usuarios que se conectan sin cuentas Shell. Gitosis es un software de

fuente abierta y se encuentra bajo las licencias de GPL.

Entre sus principales desventajas esta la falta de un control de acceso a nivel de ramas, todas las operaciones para agregar usuarios, repositorios y permisos se realizan modificando ficheros de configuración que controlan el funcionamiento de la herramienta. Esta herramienta no presenta una interfaz agradable para la administración de repositorios Git, y el mantenimiento de los usuarios, repositorios y permisos se vuelve una tarea engorrosa cuando los mismos aumentan en cantidad.

**Gitolite**

Gitolite es una herramienta para la gestión de repositorios Git basada en Gitosis y fue creada por Sitaram Chamarty. La diferencia entre Gitolite y Gitosis radica en la implementación de Gitolite, ya que esta permite realizar la administración por rama, permitiendo otorgar los permisos de escritura y lectura en dependencia de las ramas y los usuarios. Esta herramienta está escrita totalmente en Perl y diseñada para ser levantada en entornos Unix.

La herramienta Gitolite en comparación con Gitosis es una aplicación más simple pero más poderosa, cuenta con mayor cantidad de información sobre la instalación y puesta en marcha, posee una menor complejidad en la sintaxis de configuración, cuenta con facilidades para establecer los permisos por ramas a los repositorios, así como la ventaja de dividir en partes el archivo de configuración y delegar la autoridad en diferentes desarrolladores.

Independientemente de las ventajas anteriormente presentadas, al igual que Gitosis no presenta una interfaz agradable y su administración es compleja cuando los usuarios, permisos y repositorios aumentan en cantidad. Al igual que Gitosis todo el control de grupos de usuarios, repositorios y permisos se realizan directamente sobre los ficheros de configuración.

El autor de la presente investigación llegó a la conclusión de que ninguna de las herramientas estudiadas satisface las necesidades de administración de grupos de usuarios, repositorios y permisos del proyecto Nova. Tanto Redmine como Gitorious son herramientas muy completas pero carecen de una orientación a la administración de múltiples repositorios con crecientes cantidades de usuarios, repositorios y permisos. Por las ventajas que ofrece Redmine, se determinó que la aplicación a construir presente características que pudieran ser de utilidad para integrarse de manera satisfactoria en un futuro a Redmine.

La herramienta Indefero carece de un sistema de permisos a nivel de ramas y está implementado en php, motivo que provocaría dificultades para una futura integración con

Redmine. Kenai es un gestor de repositorios que al igual que Indefero es de reciente creación, y no cuenta con una amplia documentación que pudiera ser de utilidad para el desarrollo de la aplicación.

Gitosis es una herramienta orientada a la administración de repositorios pero no presenta la posibilidad de administrar los repositorios a nivel de rama. Gitolite si contiene un sistema de administración de repositorios a nivel de rama pero carece de una interfaz gráfica amigable. Por tanto, se determinó crear Gitadmin, que sería una herramienta que integraría lo mejor de la administración de repositorios Git, por lo cual se utilizaría Gitolite como medio para la administración de los repositorios Git y se le implementaría una interfaz gráfica que interactúe directamente con los ficheros de configuración y posibilite su modificación.

Capítulo

# Entorno de Desarrollo.

El entorno de desarrollo de esta investigación está compuesto por los elementos significativos utilizados para la implementación de Gitadmin. En este capítulo se analizan herramientas, tecnologías y lenguajes que por sus características e importancia, han sido seleccionados para el desarrollo de la aplicación.

La correcta selección de herramientas y tecnologías propicia un diseño y modelado apropiado de la solución. Para el modelado de Gitadmin se hace uso del lenguaje de modelado UML, mediante el UML se diseñará para Visual Estudio los principales elementos de Gitadmin, como sus clases y la interacción entre estos. Para la implementación de Gitadmin se emplea Ruby como lenguaje de programación, Ruby permitió desarrollar las principales funcionalidades de su sistema principal y mediante html se pudo visualizar las diferentes interfaces diseñadas. Para obtener un mejor entendimiento de la propuesta de solución a continuación se describe el entorno de desarrollo, el mismo está compuesto por las herramientas, tecnología, lenguajes utilizados y la metodología de investigación a usar para la implementación de la herramienta que administrará los repositorios Git.

**Lenguajes**

En el caso particular de esta investigación el lenguaje es la forma mediante la cual se establece la comunicación entre máquinas y humanos. Estos lenguajes sirven para establecer diferentes comunicaciones, y mediante estas comunicaciones las máquinas realizan las tareas que se le asignan, visualizan o manejan los objetos de acuerdo a como se haya establecido mediante el lenguaje.

**UML**

El Lenguaje Unificado de Modelado (UML) es un lenguaje estándar de modelado introducido

en la informática en el año 1997 que sirve para escribir los planos del software, el UML sirve para especificar, visualizar, construir y documentar los artefactos de los sistemas software, así como para el modelado del negocio y otros sistemas no software [5]. El UML puede usarse para modelar desde sistemas de información hasta aplicaciones distribuidas basadas en Web.

Debido a su estandarización y su definición, aunque no sea un lenguaje de programación puede ser conectado de manera directa a lenguajes de programación como Java, C++ o Visual Basic, mediante esta ingeniería directa se puede obtener el código fuente partiendo de los modelos, además es posible reconstruir un modelo en UML partiendo de la implementación ( la ingeniería inversa).

Este lenguaje tiene la capacidad de modelar actividades de planificación de proyectos y de sus versiones, expresar requisitos y las pruebas sobre el sistema, representar todos sus detalles así como la propia arquitectura. Mediante estas capacidades se obtiene una documentación que es válida durante todo el ciclo de vida de un proyecto [3].

Para la modelación de la solución este lenguaje ayuda a interpretar grandes sistemas mediante gráficos o texto obteniendo modelos que ayudan a la comunicación durante el desarrollo, estos modelos podrán ser interpretados por personas o herramientas, sin necesidad que las personas hayan participado en su diseño.

**HTML**

HyperText Markup Language (HTML) es un lenguaje de marcado con el que se definen las páginas web. Este lenguaje se usa para describir la estructura, el contenido y complementar con imágenes el texto que aparece en las páginas web. Este lenguaje surgió en un principio solo para divulgación de documentos con textos e imágenes, pero con el paso del tiempo se empezó a utilizar con propósitos de ocio y consultas especializadas con uso de multimedia y lenguajes como Javascript para aportarle dinamismo y estructura a la información [6].

Con el desarrollo de internet y los navegadores web, el HTML se convirtió en uno de los eslabones fundamentales para la elaboración de documentos para la web. La capacidad del HTML se multiplica cuando se usa en conjunto de lenguajes de programación, en la aplicación a desarrollar específicamente se implementa su uso conjuntamente con Ruby, quien ofrece funcionalidades que le aportan potencia y personalización a la aplicación que se implementara.

Este lenguaje puede ser editado por cualquier editor común como el bloc de notas de Windows o gedit de GNU/Linux; existen algunos editores especializados como BlueFish para Linux o DreamWeaver para Windows que ofrecen completamiento de etiquetas y

opciones para el desarrollo como creación automática de tablas y tipos de letras sin necesidad de poseer algún conocimiento de HTML.

En la construcción de la aplicación informática presentada como propuesta en esta investigación se hará uso del HTML en la construcción y presentación de toda la información que observará el usuario. El HTML permitirá construir una estructura organizada de la información existente, así como organizar la información insertada por el usuario.

**Ruby**

Teniendo en cuenta como nuevo objetivo de que la herramienta producto de esta investigación posea características de ventaja para realizar una integración a Redmine en un futuro, se decidió realizar la aplicación en Ruby, facilitando de esta manera la tarea de integración debido a que la herramienta Redmine está desarrollada en el lenguaje de programación Ruby. Este lenguaje cuenta con **API** para el trabajo con repositorios Git y frameworks para desarrollar aplicaciones web de diversa complejidad, entre los que tenemos Ruby On Rails, Merb y Sinatra.

A continuación se muestran algunas características de este lenguaje:

- Ruby es un lenguaje de scripts, orientado a objetos y moderno, que combina flexibilidad con potencia.
- Tiene entre sus ventajas las mejores características de otros lenguajes de programación como Small-talk, Java y Perl.
- Su alcance parece ilimitado y hoy se encuentra presente en aplicaciones que van desde el desarrollo web hasta la simulación de ambientes complejos.
- Es un lenguaje multiplataforma que tiene perfecta integración con gran cantidad de arquitecturas; puede ser usado con integración a dispositivos móviles.
- Promueve las mejores prácticas de programación sin perder usabilidad.
- Mediante su uso se pueden complementar las características de la lógica imperativa con la lógica funcional.
- Es altamente extensible, no sólo mediante librerías escritas en Ruby, sino que podemos ampliarlo utilizando el lenguaje C y, actualmente, de forma experimental otros lenguajes.
- Simplifica declaraciones, estructuras y modelos sin perder potencia y permite que el programador, se desarrolle de forma adecuada.
- Es un lenguaje dinámico e interpretado, con las características de éstos.
- Permite utilizar la más simple expresión para un programa o algoritmo; esto sumado a las actuales prácticas ágiles permite desarrollar en forma amigable.

Este lenguaje fue creado en Japón, su creador Yukihiro Matsumoto, lo desarrollo mientras trabajaba programando en lenguajes como PHP y Perl. Matsumoto en un principio tenía como objetivo crear un Perl avanzado debido a que su principal objetivo era mejorar algunas de las particularidades de Perl. Pero en vez de esto, prefirió desarrollar un lenguaje propio a partir de lagunas características y ventajas de sus lenguajes de programación preferidos: Smalltalk, Eiffel, Lisp y Perl. Así surgió Ruby y su primera versión presentada al público fue la 0.95. Sobre la creación de Ruby expresó su creador "As an object-oriented fan for more than fifteen years, it seemed to me that OO programming was very suitable for scripting too. I did some research on the net for a while, but the candidates I found, Perl and Python, were not exactly what I was looking for. I wanted a language more powerful than Perl and more object-oriented than Python."[2].

Independientemente de las características anteriormente planteadas el lenguaje Ruby cuenta con algunas particularidades:

En Ruby todo es un objeto. Esto significa que el más simple carácter o un conjunto de instrucciones, son instancias de clases y serán manipuladas como tales. Este concepto anula lo que normalmente denominamos tipos primitivos, ya que hasta el más trivial de los datos es un objeto.

La gran flexibilidad de Ruby permite la reprogramación o inclusión de nuevas funcionalidades en sus clases base y en sus métodos. Esto quiere decir que se puede modificar todo dentro del ambiente de Ruby.

Ruby utiliza solamente herencia simple. Esta característica en muchos lenguajes facilita el trabajo con estructuras jerárquicas. Sin embargo, incorpora técnicas para poder imitar el comportamiento de la herencia múltiple de manera más sencilla.

Ruby no es estricto y no requiere declaración de variables.

El lenguaje permite la programación con múltiples hilos de forma independiente al sistema operativo.

Ruby utiliza un recolector de basura de alto nivel y libera al desarrollador de estas tareas.

Actualmente este lenguaje cuenta con proyectos y soporte para arquitecturas o entornos de importancia, en estos momentos se encuentran los proyectos Ruby.net y JRuby, cada uno busca obtener interoperabilidad total entre plataformas, en estos momentos se pueden encontrar muchos entornos con diferentes características que permiten desarrollo de aplicaciones para Ruby como Geany o Erick; para proyectos más grandes podemos utilizar Eclipse, NetBeans o Komodo.

Este lenguaje cuenta con proyectos de gran complejidad como Basecamp, que no es más que un gestor y organizador de trabajo en equipo, Existen otros proyectos como Odeo, que es una red social para el intercambio de música, este portal esta desarrollado en Rails y cuenta con más de mil canales de música y un millón de archivos de audio, y además de

estos proyectos se encuentra RubyForge, un repositorio de código para Ruby que cuentan con más de diez mil usuarios.
Para el desarrollo de herramientas de administración de repositorios Git el lenguaje Ruby es uno de los más utilizados. Entre los proyectos más destacados se encuentran Redmine y Gitorious, y algunas API para Git como Ruby/git, GitStore, GitRb y Grit que es una librería para la extracción de información de los repositorios Git.

### Javascript

Javascript es un lenguaje de scripting creado en 1995 por Brendan Eich de la empresa Netscape Communications Corp y Mozilla Fundation. Este lenguaje es utilizado para acceder a objetos de aplicaciones principalmente involucradas a interfaces de usuarios y páginas web. Este es un lenguaje basado en prototipos y creado con una sintaxis análoga a Java, aunque en comparación con este último es mucho más fácil de usar [7].
Este lenguaje no requiere de tiempo de compilación ya que es un lenguaje interpretado. Es un lenguaje seguro ya que tiene acceso mínimo al cliente una vez descargado el código a interpretar, aunque es recomendado utilizarlo en conjunto con otros lenguajes. Es un lenguaje flexible que permite realizar las operaciones en tiempo de ejecución.
En el presente trabajo es utilizado para la presentación de las interfaces de usuario en conjunto con el código HTML. Este lenguaje se ejecuta del lado del cliente, y es compatible con la mayoría de los navegadores web.

## Tecnología

### Ruby-Gems

En la actualidad existe la fuerte tendencia a la reutilización de código e integración entre herramientas, tecnologías y lenguajes de programación; los lenguajes deben proveer herramientas y características que facilite el trabajo. Ruby, siguiendo con la filosofía de trabajo “No te repitas”, expresado en inglés “Don't repeat yourself” (DRY) se encuentra integrado a la perfección con las tecnologías que actualmente se encuentran en uso como bases de datos, XML, HTML y distribución de paquetes entre otras. Una de las tecnologías más usada por los usuarios de Ruby son las Gemas de Ruby o Ruby-Gems; Ruby utiliza un gestor de paquetes llamado RubyGems, este proporciona un formato estándar (.gem) con el objetivo de distribuir programas o librerías que sean capaces de incorporar o quitar funcionalidades al sistema de ruby que se desarrolla. Este cuenta con herramientas para

gestionar la instalación y servidor para la distribución. Entre las principales funcionalidades se destacan:

- Instalar los paquetes a distancia.
- Administrar a distancia.
- Administrar dependencias.
- Desinstalar de forma fácil.

**Metodología de Desarrollo**

Para la producción de software de calidad en la actualidad, es prácticamente imposible desarrollar aplicaciones sin seguir las pautas de alguna metodología de desarrollo de software. La metodología constituye la columna vertebral en un proceso de desarrollo de software, las metodologías definen quien debe hacer que, cuando y como debe hacerlo. Estas metodologías difieren en sus características, basándose en las necesidades reales de los desarrolladores y clientes, del tiempo y recursos con los que se dispone, y principalmente del grado de complejidad del software y predictibilidad que se desea en el desarrollo de software [8].

Para la elaboración de la herramienta se hace necesario el uso de una metodología de desarrollo de software ágil. El uso de una metodología ágil permite centrarse en los individuos y las iteraciones por encima de los procesos y las herramientas. Esta metodología no requiere de la generación de una documentación exhaustiva. Requiere de una estrecha comunicación con el cliente. Para esta metodología es más importante la respuesta ante el cambio que el seguimiento de un plan, esta se adapta con facilidad a los cambios de requisitos, tecnología o equipamiento.

**Programación Extrema (XP o eXtreme Programing)**

La programación extrema nació en la comunidad de Smalltalk, y su auge se debió en gran medida a la colaboración entre Ward Cunningham y Kent Beck, esta metodología se fue mejorando mediante su uso en algunos pequeños proyectos a inicios de los 90. XP es una de las metodologías ágiles más populares en la actualidad para proyectos de pocos desarrolladores y con cortos períodos de tiempo para desarrollar. La Programación Extrema es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado [9]. Esta metodología presenta gran flexibilidad y capacidad para adaptarse a los cambios en los requerimientos, estos requerimientos y las prioridades se pueden reajustar durante el proyecto en periodos cortos y regulares de tiempo. Sobre la concepción de esta

metodología uno de sus creadores Kent Beck expresó "Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar."[1]. La cita anterior refleja una de las características fundamentales de la metodología XP, esta fue una de las razones por lo cual se eligió XP como metodología de desarrollo, esta característica es la facilidad con que cuenta esta metodología para adaptarse a los cambios de requisitos y tecnologías.

Características fundamentales de esta metodología:

- Desarrollo iterativo e incremental.
- Cuenta con una fuerte presencia del usuario final en el equipo de desarrollo.
- Pruebas unitarias continuas y automatizadas.
- Refactorización del código. Es la reedición del código con el objetivo de permitir una mejor lectura del mismo, optimización.
- Código compartido. Esta forma de trabajo aboga porque cualquier desarrollador tenga la potestad para modificar y trabajar en cualquier parte del proyecto, y se haga responsable por los cambios realizados.
- Simplicidad del código. Esta estrategia de trabajo permite que cada parte del proyecto sea fácil de modificar o al menos que no se apliquen códigos complejos, engorrosos y duplicados.
- Releases pequeñas. Se intenta ir liberando pequeñas partes funcionales del proyecto que resulten de valor para el cliente.

**Herramientas**

**Sinatra**

Sinatra es un framework para el lenguaje Ruby, un framework es un conjunto de librerías y componentes que permiten realizar el diseño, la construcción y puesta en marcha de aplicaciones de manera más rápida y de mayor calidad. Los frameworks presentan a los desarrolladores variadas facilidades para el desarrollo de aplicaciones, así como sistemas de manejo de url, capas de acceso a datos, facilidades de configuración de puertos y personalización.

Teniendo en cuenta que el lenguaje a utilizar es Ruby, se determinó que se utilizaría el framework Sinatra, teniendo en cuenta principalmente su simpleza y su capacidad de adaptación ante proyectos de corto, medio y largo alcance tanto de tiempo como de

complejidad. Sinatra se usa principalmente en la creación de interfaces rápidas para códigos de prueba y conexión visual a núcleos o sistemas de clases. Este framework permite separar las vistas del código de la aplicación y de la capa de acceso a datos debido a que es de tipo VC (Vista, Controlador).

Para el desarrollo de la herramienta mediante Sinatra la capa Vista será la que presenta la interfaz que interactúa directamente con el usuario, presenta estilos de letras y estilos definidos de presentación. La capa Controlador contiene toda la lógica de la aplicación, desde esta capa se manejan las funciones vitales de la herramienta que permiten un correcto funcionamiento.

Este framework posee facilidades para la puesta en marcha de aplicaciones sin grandes complicaciones y no requiere de grandes tareas de mantenimiento. Posee gran flexibilidad, esta característica permite adaptarle con facilidad la API para el manejo de repositorios Git y la adición de gemas de ruby para facilitar el trabajo con librerías.

**Geany**

Geany es un IDE (Integrated Development Environment) de código abierto muy pequeño y ligero. Posee entre sus ventajas resaltado y colapsado de código, autocompletado de código, identificación de XML y HTML, atajos de teclado, posibilidad de deshacer/rehacer, sistema de pestañas y soporta los lenguajes C, Java, PHP, HTML, Python, Perl, Pascal y Ruby. Esta herramienta cuenta con la ventaja de ser multiplataforma. Geany posee un intérprete para el código y en la barra izquierda representa las estructuras y clases que aparecen en él, además posee la ventaja de ampliar sus funcionalidades mediante plugins y complementos. Esta aplicación llama al compilador y luego ejecuta el programa compilado directamente a través de una consola que se integra en el programa.

**Visual Paradigm**

Visual Paradigm es una herramienta que soporta el ciclo de vida completo del desarrollo de software. Este contempla fases de análisis y diseño orientado a objetos, construcción, pruebas y despliegue. El modelado de software mediante Visual Paradigm utilizando UML permite construir aplicaciones de calidad en un tiempo relativamente corto.

Esta herramienta permite producir la documentación del sistema usando formato PDF y HTML. Los desarrolladores pueden producir la documentación sobre el desarrollo del software en cuestión usando plantillas predeterminadas. La herramienta Visual Paradigm permite exportar código Java a partir del diseño de clases mediante UML y está especializada en la ingeniería del software para bases de datos.

En el caso particular de esta investigación Visual Paradigm se usa con el objetivo de auxiliarnos en el diseño de algunos artefactos generados a partir de las tareas de ingeniería, como es el caso del diagrama de dominio y los diagramas de clases.

Capítulo

# Descripción y Desarrollo de la Solución Propuesta.

## Descripción de la solución propuesta

La implementación de Gitadmin se fraccionó según la metodología XP en diferentes historias de usuarios. Estas historias de usuarios se desarrollan en un proceso iterativo e incremental y que mediante tres iteraciones permitirían obtener en cada una de ellas como resultado la aplicación pero con un conjunto limitado de funcionalidades acabadas. La culminación de las iteraciones daría como resultado una versión terminada de Gitadmin.

La aplicación a construir estará compuesta por dos piezas fundamentales, la primera pieza es la API encargada del manejo de los grupos de repositorios Git, usuarios y permisos. Esta API debe ser capaz de interpretar la configuración sobre los repositorios Git guardados. La API contiene todas las funcionalidades necesarias para insertar, modificar o eliminar los grupos de repositorios, grupos de usuarios y permisos. La segunda pieza es la implementación de un sistema de vistas para interactuar con las funcionalidades que ofrece la API. El manejo de estas vistas se llevara a cabo mediante Sinatra.

## Descripción de la arquitectura cliente-servidor

Para el desarrollo de Gitadmin se utiliza la arquitectura cliente-servidor. Esta arquitectura utiliza un conjunto de estándares, reglas y procesos, que permite integrar un amplio conjunto de aplicaciones informáticas. Esta arquitectura debe su nombre a que la lógica de su funcionamiento está compuesta por un cliente que es el encargado de realizar las peticiones y órdenes que debe realizar el servidor. El servidor subordinándose al diseño e implementación de su programación interna resuelve y responde a las peticiones del cliente.

Esta arquitectura responde a las necesidades de esta investigación permitiendo que un servidor (en el caso particular de esta investigación el servidor contiene la API, para

responder las peticiones) pueda responder peticiones de múltiples clientes (la interfaz de usuario) concurrentemente. Además, permite realizar cambios en la programación del servidor de manera que sea transparente al cliente. Esta arquitectura permite que se realicen actualizaciones o remplazo tecnológico sin que influya directamente en el usuario. La arquitectura cliente-servidor se caracteriza por ser flexible y escalable permitiendo en un futuro realizar cambios y modificaciones de forma sencilla.

**Asimilación de Gitolite y descripción de elementos de utilidad**

El proceso de administración de Gitadmin se verá auxiliado por Gitolite. Gitolite contiene los ficheros de configuración de administración de los repositorios, las llaves públicas de acceso de los usuarios y los repositorios. En la construcción de una aplicación para la administración de los repositorios Git en Nova se hará uso del fichero de configuración que administra los grupos de usuarios y repositorios, y contiene los permisos, y se trabajará con el archivo que contiene las llaves públicas de los usuarios con acceso a los repositorios.

El fichero de configuración perteneciente a Gitolite encargado de manejar los grupos de usuarios, repositorios y permisos se llama “gitolite.conf”, este se encuentra en el archivo “conf”. Este fichero contiene toda la configuración global referente a la administración del acceso a repositorios y el acceso a ramas particulares por usuarios o grupos determinados.

El fichero contiene la siguiente estructura para declarar un grupo de usuarios (Ejemplo 1). Para que el sistema interprete un grupo de usuarios es necesario que todos los usuarios posean una llave de acceso o de lo contrario no se tomará como un grupo válido. Los grupos de usuarios tienen la posibilidad de contener otros grupos de usuarios declarados anteriormente (Ejemplo 2).

Ejemplo 1) @grupo1 = usuario1 usuario2

Ejemplo 2) @grupo2 = @grupo1 usuario3 usuario4

Para la declaración de los grupos de repositorios se sigue la lógica aplicada a los grupos de usuario, donde los repositorios pueden contar con una cantidad indefinida de repositorios (Ejemplo 3). Estos cuentan con la posibilidad al igual que los grupos de usuarios de contener otros grupos declarados con anterioridad (Ejemplo 4).

Ejemplo 3) @repo1 = desarrollo prueba gentoo

Ejemplo 4) @repo2 = @repo1 experimental

La declaración de los permisos se realiza incluyendo al principio de la línea la palabra repo y seguido una lista de repositorios que puede incluir un grupo de repositorios. Esta declaración va seguido en la(s) línea(s) próxima(s) por el tipo de acceso (R,RW,R+,RW+) que determina si es de lectura o escritura el acceso de la lista de usuario (Ejemplo 5). Los permisos pueden ser asignados directamente a una rama específica (Ejemplo 6).

Ejemplo 5) repo gentoo desarrollo @repo2

R = @grupo1 usuario3

RW = usuario4 usuario1

Ejemplo 6) repo experimental

R nueva_rama = usuario3 usuario1

RW rama1 = usuario4 @grupo1

Este fichero debe ser interpretado por la API de Gitadmin, la cual debe ser capaz de diferenciar grupos de usuarios y repositorios, así como permisos. Con la debida interpretación de este fichero de configuración podrá realizar el proceso de administración para gestionar grupos de usuarios, repositorios y permisos.

Un archivo significativo para la aplicación que se persigue construir con esta investigación es “keydir”, este es el encargado de guardar las llaves públicas de acceso de los diferentes usuarios del sistema, estos pueden tener o no un nivel de administración, que se encuentren en esta carpeta no quiere decir específicamente que administra un repositorio. El nombre de la llave de un usuario está compuesto por el nombre del usuario y una extensión “.pub”. La API a desarrollar realizará una lectura de las llaves públicas que se encuentran el archivo “keydir”, de aquí podrá determinar si un grupo pertenece a usuarios o repositorios. Para que un grupo de usuarios sea declarado como tal todos los usuarios deben tener una llave pública en el archivo que guarda las llaves de los usuarios, de lo contrario se declarará como un grupo de repositorios.

**Desarrollo de la herramienta**

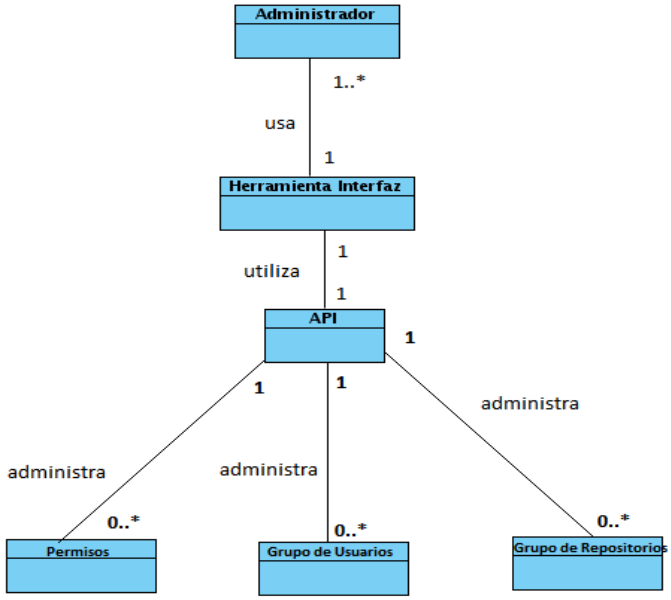
El desarrollo de la herramienta gráfica Gitadmin constituye la interfaz para el usuario de la herramienta anteriormente analizada Gitolite. Gitadmin es la encargada de ofrecer las funcionalidades disponibles al usuario mediante una interfaz simple.

**Modelo de Dominio**

El Modelo de Dominio o Modelo Conceptual es la representación visual de los elementos significativos del mundo real y sus interacciones para resolver un problema específico. En la presente investigación el Modelo de Dominio es utilizado como una herramienta de comunicación para permitir un mejor entendimiento de la interacción entre los diferentes sistemas que interactuaran en la herramienta.

La herramienta interfaz constituiría la presentación web de Gitadmin, mediante esta interfaz se tendrá acceso a todos los métodos de administración desarrollados sobre la API. La interfaz de Gitadmin debe ser capaz de administrar los grupos de usuarios con acceso al

sistema, los grupos de repositorios y los permisos con su sistema de acceso a nivel de ramas.




**Descripción de la API**

La API contiene un conjunto de clases capaces de manejar los datos concernientes a los grupos de repositorios, grupos de usuarios y permisos. De los permisos debe ser capaz de asimilar las diferentes ramas, en caso de llevar ramas y los usuarios o grupos de usuario con acceso a la misma. Esta API es capaz de interactuar con el fichero de configuración y obtener mayor provecho de la misma mediante la no inclusión de errores en la definición de los grupos de usuarios y permisos, y su posterior depuración mediante la exportación de un nuevo fichero de configuración.

Entre las principales funcionalidades se encuentra el manejo a nivel de usuario y repositorios las características de los mismos (nombre y descripción). Para los permisos esta API es capaz de manejar sus diferentes ramas, permisos de acceso (escritura y lectura) y grupos de usuarios o usuarios independientes. Para la administración de grupos de repositorios y grupos de usuarios, contiene facilidades que permiten listar, insertar, modificar y eliminar como funcionalidades básicas.

Esta API se hizo con el objetivo de compatibilizar las características de administración de Gitolite con Redmine, pero puede ser usada desde cualquier aplicación que se desee. Para obtener las características de administración de Gitolite y adaptarle un sistema de interfaces para cualquier sistema solo es necesario contar con la herramienta Gitolite y la API de Gitadmin, e incluir esta última desde la aplicación que se desee manejar los repositorios Git.

**Definición de las historias de usuarios**

Para el desarrollo de la herramienta que permitirá la administración de grupos de repositorios Git, usuarios y permisos se realizó la definición de las historias de usuarios. Estas historias de usuario constituyen una guía durante todo el período que dure la implementación de la herramienta.

Historias de usuario a implementar:

- Listar grupos de usuarios.
- Insertar grupo de usuarios.
- Modificar grupos de usuarios.
- Eliminar grupos de usuarios.
- Listar grupos de repositorios.
- Insertar grupo de repositorios.
- Modificar grupo de repositorios.
- Eliminar grupo de repositorios.
- Listar permisos.
- Insertar permiso.
- Modificar permiso.
- Eliminar permiso.
- Guardar configuración.

**Planificación de historias de usuarios**

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 muestra mediante una tabla la planificación de las diferentes historias de usuario para cada iteración teniendo en cuenta su prioridad:

| No | Nombre de Historia de Usuario | Prioridad | Riesgo | Iteración |
|---|---|---|---|---|
| 1 | Listar grupos de usuarios | Baja | Bajo | 3 |
| 2 | Insertar grupo de usuarios | Media | Bajo | 2 |

| | | | | |
|---|---|---|---|---|
| 3 | Modificar grupo de usuarios | Alta | Alto | 1 |
| 4 | Eliminar grupos de usuarios | Baja | Bajo | 2 |
| 5 | Listar grupos de repositorios | Baja | Bajo | 3 |
| 6 | Insertar grupo de repositorios | Media | Medio | 2 |
| 7 | Modificar grupo de repositorios | Alta | Medio | 1 |
| 8 | Eliminar grupo de repositorios | Media | Bajo | 3 |
| 9 | Listar permisos | Baja | Bajo | 3 |
| 10 | Insertar permiso | Media | Medio | 2 |
| 11 | Modificar permiso | Alta | Alto | 1 |
| 12 | Eliminar permiso | Media | Medio | 2 |
| 13 | Guardar configuración | Alta | Medio | 2 |

Al finalizar cada iteración se obtiene como resultado un paquete de funcionalidades de acuerdo a la prioridad definida anteriormente. El primer paquete de funcionalidades correspondiente a la primera iteración corresponde a las historias de usuarios encargadas de la modificación del sistema, estas son las historias de usuario #3, #7 y #11. Para la segunda iteración se obtendrían las funcionalidades correspondientes a las historias de usuario #2, #4, #6, #10, #12, #13. La historia de usuario #13 encargada de guardar la configuración se dispuso para la segunda iteración ya que dependía de las funcionalidades de la primera iteración para que se pudiera llevar a cabo. El resto de las funcionalidades quedaron sujetas a la tercera iteración.

**Modelación de historias de usuario**

**Historias de usuario**

Para dar solución a la problemática existente se desarrollaron las historias de usuario que permitirían dar solución a las necesidades existentes. Estas historias de usuario se fueron agrupando de acuerdo a la iteración en la que fueron desarrolladas, teniendo en cuenta como anteriormente se planteo su prioridad. Las tareas de ingeniería aplicadas a cada historia de usuario se muestran a continuación de las mismas, los tipos de tareas se clasifican en Desarrollo, Corrección y Mejora. Seguido de las tareas de ingeniería se plantean las pruebas de aceptación que se determinaron para cada historia de usuario,

estas ayudan a que cada iteración devuelva la aplicación a desarrollar con un subconjunto de funcionalidades sometidas a prueba y aprobadas. Los diagramas de clases generados a partir de las tareas de ingeniería aplicadas a cada historia de usuario pueden ser observados en el anexo 3.

- **Iteración 1**

- ✓ **Historia de Usuario 3**

| Historia de Usuario | |
|---|---|
| No: 3 | Nombre: Modificar un grupo de usuarios. |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Alto |
| Descripción: Muestra la lista de grupos de usuarios disponibles, se escoge uno de ellos y se modifica. | |
| Observaciones | |

Mediante la interfaz que se muestra en la **Fig. 1** se realiza la selección del grupo de usuario que se desea modificar, en esta lista se muestran los grupos de usuarios que el sistema tiene disponible.


**Fig. 1. Seleccionar grupo de usuarios.**

Después de seleccionado el grupo de usuario se procede a su modificación **Fig. 2**.

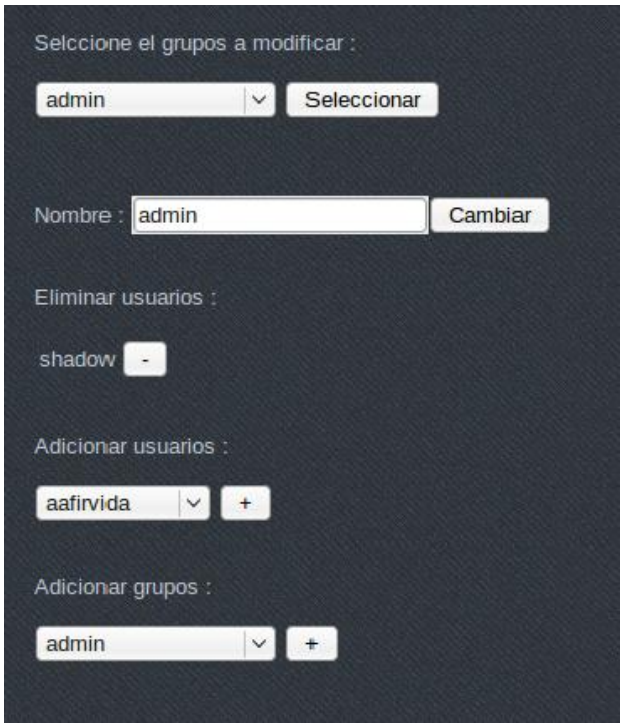



**Fig. 2. Modificar grupo de usuarios.**

Mediante la interfaz mostrada en la **Fig. 2** se puede modificar para un grupo de usuario el nombre, adicionar usuarios o grupos de usuarios o eliminar usuarios existentes.

Para dar cumplimiento a la anterior historia de usuario se definió una tarea de ingeniería que consiste en la construcción de una clase que garantizará las funcionalidades básicas de modificación de los grupos de usuarios:

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 1 | Número Historia de Usuario: historia de usuario 3 |
| Nombre Tarea: Programar clase Group_User. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 7/02/2010 | Fecha Fin: 10/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Clase encargada de manejar los grupos de usuarios. | |

Esta clase lee el fichero de configuración de Gitolite “gitolite.conf” y partir de este fichero maneja los diferentes atributos que tendrá un grupo de usuarios determinado.

- ✓ **Historia de Usuario 7**

| Historia de Usuario | |
|---|---|
| No: 7 | Nombre: Modificar grupo de repositorios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Medio |
| Descripción: Muestra la configuración que tiene actualmente el grupo de repositorios, permite modificar la configuración existente o insertar nuevo repositorio o grupo de repositorios. | |
| Observaciones: | |

Esta historia de usuario contiene una tarea de ingeniería, esta es la encargada de administrar las características y funcionalidades para los grupos de repositorios:

| Tarea de Ingeniería | | |
|---|---|---|
| Número Tarea: 2 | Número Historia de Usuario: historia de usuario 7 | |
| Nombre Tarea: Programar clase Group_Repo. | | |
| Tipo de Tarea: Desarrollo. | | |
| Fecha Inicio: 11/02/2010 | | Fecha Fin: 12/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | | |
| Descripción: Clase encargada de manejar los grupos de repositorios. | | |

- ✓ **Historia de usuario 11**

| Historia de Usuario | |
|---|---|
| No: 11 | Nombre: Modificar permisos |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Alto |
| Descripción: Muestra la lista de permisos Fig. 3, se escoge el permiso a modificar y obtenemos la configuración del permiso Fig. 4, podemos modificar los permisos existentes o insertar nuevos permisos. | |
| Observaciones: | |


**Fig. 3. Seleccionar permiso.**

En la **Fig. 3** se muestra la interfaz que genera el listado de permisos disponibles del sistema para modificar. Después de seleccionado el permiso se procede a modificar el mismo Fig. 4.


**Fig. 4. Modificar permiso.**

Los permisos se modifican orientados al sistema de permisos de Gitolite, teniendo en cuenta esto, se diseño una interfaz para modificar las líneas que componen un permiso determinado.

Para dar cumplimiento a la historia de usuario 11 se designó una tarea de ingeniería:

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 3 | Número Historia de Usuario: historia de usuario 11 |
| Nombre Tarea: Programar clase Permisos. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 14/02/2010 | Fecha Fin: 16/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Clase encargada de manejar los Permisos. | |

**Casos de prueba para la iteración 1**

Los casos de prueba son la planificación de los escenarios mediante los cuales se comprobará el correcto funcionamiento de determinadas historias de usuarios. En su gran mayoría, los escenarios de prueba que aquí se muestran están destinados a la correcta inserción de algunos elementos que en caso de insertarse incorrectamente en el sistema pudieran ocasionar daños en la producción de software en el proyecto Nova. A continuación se muestran los escenarios de prueba para la iteración 1:

| Escenario 1. Elegir grupo de usuario a modificar | HU : 3 |
|---|---|
| Descripción:<br>El usuario escoge la opción de modificar grupos de usuarios. | Resultados esperados:<br>Muestra la actual modificación del grupo de usuarios seleccionado.<br>Muestra los campos de texto editables para el grupo de usuarios seleccionado. |
| Resultados obtenidos :<br>Correcto. | |
| Observaciones: | |

| Escenario 2. Modificar campos de texto del grupo de usuarios seleccionado | HU : 3 |
|---|---|
| Descripción:<br>El usuario modifica los campos editables del grupo de usuarios. | Resultados esperados:<br>Guarde correctamente los cambios realizados al grupo de usuarios.<br>Muestra un cartel de modificación de grupo de usuarios realizada correctamente. |
| Resultados obtenidos:<br>Muestra un cartel de se modificó correctamente el grupo de usuarios.<br>Correcto. | |
| Observaciones:<br>Un grupo de usuarios no puede quedarse sin ningún usuario. En caso de que se intente eliminar todos los usuarios de un grupo este no lo permitirá. En los archivos de configuración no se guardan grupos de usuarios vacios. | |

| Escenario 1. Modificar un grupo de repositorios | HU : 7 |
|---|---|
| Descripción:<br>El usuario escoge la opción de modificar un grupo de repositorios de un listado de repositorio. El usuario modifica la configuración existente o agrega nuevas configuraciones. | Resultados esperados:<br>Muestre un listado para modificar los grupos de repositorios.<br>Muestra los campos para modificar el grupo de repositorios escogido.<br>Muestra un cartel informando del éxito de la modificación. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el grupo de usuarios fue modificado correctamente.<br>Correcto. | |
| Observaciones:<br>En caso de que se deje una configuración en blanco debe mostrarse un mensaje de error. | |

| Escenario 1. Modificar permisos | HU : 11 |
|---|---|
| Descripción:<br>El usuario escoge la opción de modificar permisos, escoge de un listado el permiso para modificar y modifica la configuración existente o agrega nueva configuración. | Resultados esperados:<br>El sistema muestra un listado de permisos.<br>Muestre los campos con la configuración actual del permiso y campos para agregar nueva configuración.<br>Muestre un cartel de éxito después de la operación. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el permiso fue modificado correctamente.<br>Correcto. | |
| Observaciones:<br>En caso de que la configuración quede en blanco mostrar un cartel de error. | |

- **Iteración 2**

- ***Historia de usuario 2***

| Historia de Usuario | |
|---|---|
| No: 2 | Nombre: Insertar grupo de usuarios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Bajo |
| Descripción: Muestra los campos disponibles para insertar un grupo de usuarios, y se escogen los usuarios y grupos que pertenecerán al que grupo de usuarios que se crea Fig. 5. | |
| Observaciones: | |


**Fig. 5. Insertar grupo de usuarios.**

En la Fig. 5 se muestra la interfaz para adicionar nuevos grupos de usuarios.

Para la implementación de la historia de usuario 2 se definió la tarea de ingeniería encargada de manejar el nombre y una descripción:

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 4 | Número Historia de Usuario: historia de usuario 2 |
| Nombre Tarea: Programar clase Usuario. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 15/02/2010 | Fecha Fin: 16/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Clase encargada de manejar los atributos del usuario, del mismo se guarda el nombre y una descripción. | |

- ✓ **Historia de usuario 4**

| Historia de Usuario | |
|---|---|
| No: 4 | Nombre: Eliminar grupo de usuarios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Bajo |
| Descripción: Muestra la lista de grupos de usuarios existentes Fig. 6, de esa lista se elimina el grupo deseado. | |
| Observaciones | |

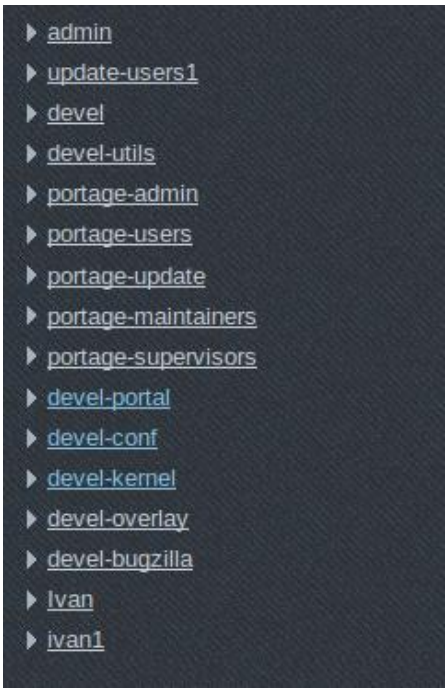



**Fig. 6. Eliminar grupos de usuarios.**

Para eliminar un grupo de usuarios solo necesita seleccionarlo. Aun después de eliminado el grupo de usuarios del sistema con el cual se interactúa, los cambios no se aplican hasta que se pone en práctica la historia de usuario #13 encargada de aplicar cambios.

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 5 | Número Historia de Usuario: historia de usuario 4 |
| Nombre Tarea: Programar clase Config. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 18/02/2010 | Fecha Fin: 22/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Clase encargada de manejar la configuración general de Gitolite, esta se encarga de almacenar los grupos de usuarios, repositorios y permisos, así como el acceso a nivel de ramas a los repositorios. | |

- ✓ **Historia de usuario 6**

| Historia de Usuario | |
|---|---|
| No: 6 | Nombre: Insertar grupo de repositorios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Medio |
| Descripción: Muestra los campos para insertar un grupo de repositorios. | |
| Observaciones: Los grupos de repositorios pueden estar compuestos por otros grupos de repositorios. | |

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 6 | Número Historia de Usuario: historia de usuario 6 |
| Nombre Tarea: Adicionar a la clase Config la funcionalidad de ingresar un nuevo grupo de repositorios. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 23/02/2010 | Fecha Fin: 24/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad para insertar un nuevo grupo de repositorios. | |

- ✓ **Historia de usuario 10**

| Historia de Usuario | |
|---|---|
| No: 10 | Nombre: Insertar permiso |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Medio |
| Descripción: Muestra los campos para insertar los repositorios y las líneas de permisos. | |
| Observaciones: | |

Con la ejecución de la siguiente tarea de ingeniería el sistema debe ser capaz de convertir el formato de permisos de los usuarios para acceder a los repositorios y sus ramas al formato definido anteriormente por Gitolite:

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 7 | Número Historia de Usuario: historia de usuario 10 |
| Nombre Tarea: Adicionar funcionalidad agregar permiso. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 25/02/2010 | Fecha Fin: 26/02/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad que manejara el ingreso de nuevos permisos con sus atributos. | |

- ✓ **Historia de usuario 12**

| Historia de Usuario | |
|---|---|
| No: 12 | Nombre: Eliminar permisos |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Medio |
| Descripción: Muestra la lista de permisos y se selecciona el permiso a eliminar. | |
| Observaciones: | |

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 8 | Número Historia de Usuario: historia de usuario 12 |
| Nombre Tarea: Programar funcionalidad eliminar permiso. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 2/03/2010 | Fecha Fin: 3/03/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad encargada de eliminar los permisos por su identificador. | |

- ✓ **Historia de usuario 13**

| Historia de Usuario | |
|---|---|
| No: 13 | Nombre: Guardar Configuración |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Medio |
| Descripción: Guarda los cambios después de haber modificado los grupos de usuarios, repositorios o permisos. | |
| Observaciones: | |

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 9 | Número Historia de Usuario: historia de usuario 13 |
| Nombre Tarea: Programar funcionalidad guardar configuración. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 4/03/2010 | Fecha Fin: 6/03/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad encargada de guardar al fichero “gitolite.conf” los cambios en la configuración general de Gitolite. | |

Esta tarea de ingeniería convierte los cambios realizados y las configuraciones insertadas a la configuración de Gitolite en una estructura entendible por Gitolite, la funcionalidad de guardar los cambios modifica el fichero “gitolite.conf”.

**Casos de prueba para la iteración 2**

| Escenario 1. Elegir opción de insertar grupo de usuarios | HU : 2 |
|---|---|
| Descripción:<br>El usuario escoge la opción de insertar nuevo grupo de usuarios. | Resultados esperados:<br>Muestra los campos de texto con los parámetros de un grupo de usuarios. |
| Resultados obtenidos:<br>Correcto. | |
| Observaciones: | |

| Escenario 2. Rellenar campos para insertar grupo de usuarios | HU : 2 |
|---|---|
| Descripción:<br>El usuario llena los campos para insertar un grupo de usuarios. | Resultados esperados:<br>Que guarde los cambios del nuevo grupo de usuarios. |
| Resultados obtenidos:<br>Guarda correctamente los datos del grupo de usuarios.<br>Correcto. | |
| Observaciones:<br>En caso de que se intente guardar con todos los campos de texto vacios se queda en la página en espera de que se inserte algún dato. No se puede insertar un grupo de usuarios con solo el nombre, el grupo de usuarios debe tener al menos un usuario, en el archivo de configuración no se admiten grupos de usuarios vacios. | |

| Escenario 1. Eliminar un grupo de usuarios | HU : 4 |
|---|---|
| Descripción:<br>El usuario escoge la opción de eliminar un grupo de usuarios y el sistema te muestra un listado de grupos de usuarios del cual se eliminara el seleccionado. | Resultados esperados:<br>Elimine el grupo de usuarios seleccionado y guarde los cambios.<br>Muestra un cartel informando de que se ha eliminado correctamente el grupo de usuarios. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el grupo de usuarios fue eliminado correctamente.<br>Correcto. | |
| Observaciones: | |

| Escenario 1. Mostrar campos para insertar grupo de repositorios | HU : 6 |
|---|---|
| Descripción:<br>El usuario escoge la opción de insertar grupo de repositorios y muestra los campos de un grupo de repositorios. | Resultados esperados:<br>Muestra los campos para insertar un grupo de repositorios. |
| Resultados obtenidos:<br>Correcto. | |
| Observaciones: | |

| Escenario 2. Llenar campos de nuevo grupo de repositorios | HU : 6 |
|---|---|
| Descripción:<br>El usuario llena los campos del grupo de repositorio y este se guarda. | Resultados esperados:<br>Muestra un cartel indicando que el grupo de usuario ha sido insertado correctamente. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el grupo de usuarios fue insertado correctamente.<br>Correcto. | |
| Observaciones: En caso de que se entren los campos vacios enviar un mensaje de error indicando que se no se ha insertado el grupo de repositorios. | |

| Escenario 1. Insertar permiso | HU : 10 |
|---|---|
| Descripción:<br>El usuario escoge la opción de insertar permisos, el sistema muestra los campos para insertar permisos. | Resultados esperados:<br>El sistema muestra los campos de insertar permisos.<br>Guarda el permiso.<br>Muestra un cartel de que se insertó con éxito el permiso. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el permiso fue insertado correctamente.<br>Correcto. | |
| Observaciones:<br>En caso de que deje algún campo en blanco se debe mostrar un mensaje de error. | |

| Escenario 1. Eliminar permiso | HU : 12 |
|---|---|
| Descripción:<br>El usuario escoge la opción de eliminar permisos y se muestra un listado con los permisos actuales, se marca el permiso que se desea eliminar. | Resultados esperados:<br>El sistema muestra una lista de permisos.<br>Que se elimine el permiso seleccionado. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el permiso fue eliminado correctamente.<br>Correcto. | |
| Observaciones: | |

| Escenario 1. Guardar configuración | HU : 13 |
|---|---|
| Descripción:<br>El usuario guarda la configuración con los cambios efectuados a los grupos de usuarios, repositorios y permisos. | Resultados esperados:<br>Muestra la nueva configuración que ha sido guardada.<br>Muestra un cartel de éxito. |
| Resultados obtenidos:<br>Correcto. | |
| Observaciones: | |

- **Iteración 3**

- ✓ **Historia de usuario 1**

| Historia de Usuario | |
|---|---|
| No: 1 | Nombre: Listar grupos de usuarios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Bajo |
| Descripción: Muestra la lista de grupos de usuarios disponibles. Fig. 7. En caso de seleccionarse alguno se pasa a una interfaz que muestra los usuarios y grupos de usuarios que contienen el grupo seleccionado. | |
| Observaciones: Para que sea lea un grupo de usuario todos los usuarios deben tener llaves publicas de acceso al sistema, de lo contrario será tomado como un grupo de repositorio. | |


**Fig. 7. Listar grupos de usuarios.**

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 10 | Número Historia de Usuario: historia de usuario 1 |
| Nombre Tarea: Programar funcionalidad listar grupos de usuarios. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 7/03/2010 | Fecha Fin: 8/03/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad encargada de mostrar los grupos de usuarios existentes. | |

- ✓ **Historia de usuario 9**

| Historia de Usuario | |
|---|---|
| No: 9 | Nombre: Listar permisos |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Bajo |
| Descripción: Muestra un listado de los permisos existentes. Fig. 8. Si se selecciona un permio determinado se muestra un interfaz con las líneas de permisos existentes. | |
| Observaciones: | |

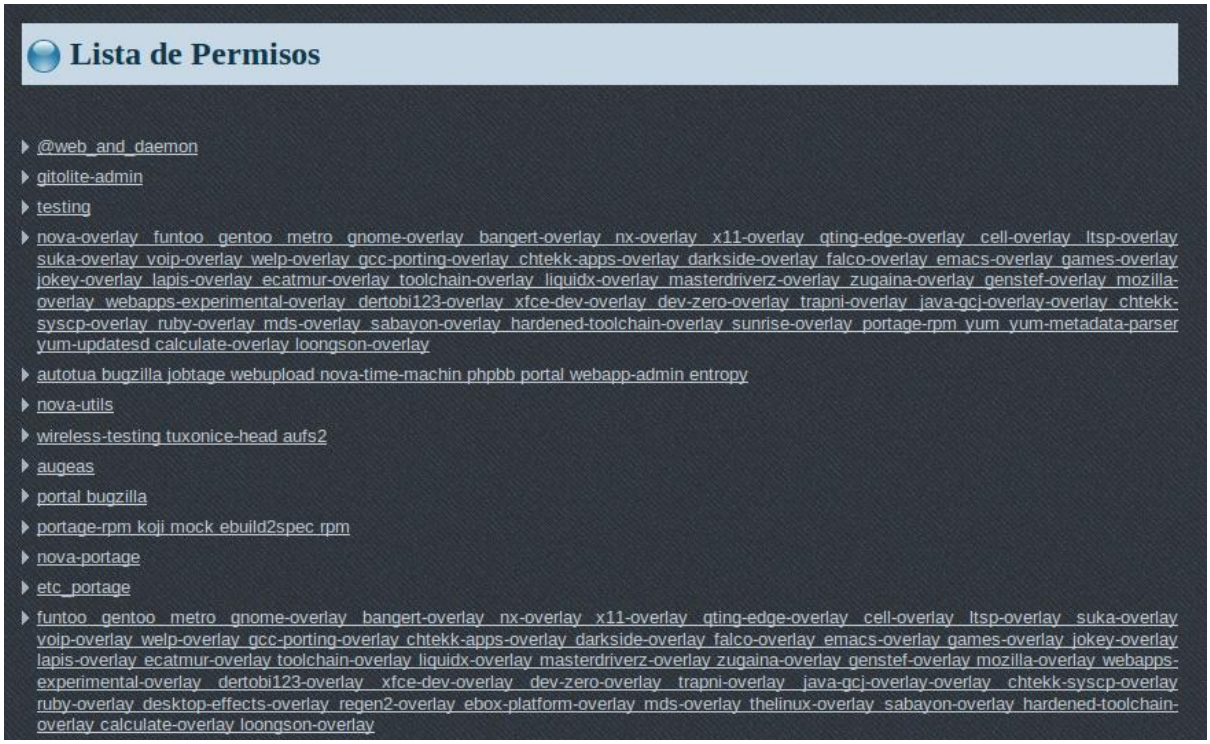



**Fig. 8. Listar permisos.**

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 11 | Número Historia de Usuario: historia de usuario 9 |
| Nombre Tarea: Programar funcionalidad Listar permisos. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 9/03/2010 | Fecha Fin: 10/03/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad encargada de generar un listado de los permisos. | |

- ✓ **Historia de usuario 5**

| Historia de Usuario | |
|---|---|
| No: 5 | Nombre: Listar grupos de repositorios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Bajo |
| Descripción: Muestra la lista de grupos de repositorios. Si se selecciona alguno muestra los repositorios contenidos dentro del mismo. | |
| Observaciones: | |

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 12 | Número Historia de Usuario: historia de usuario 5 |
| Nombre Tarea: Programar funcionalidad listar grupos de repositorios. | |
| Tipo de Tarea: Desarrollo. | |
| Fecha Inicio: 12/03/2010 | Fecha Fin: 13/03/2010 |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad encargada de generar toda la lista de grupos de repositorios. | |

- ✓ **Historia de usuario 8**

| Historia de Usuario | |
|---|---|
| No: 8 | Nombre: Eliminar grupo de repositorios |
| Usuario: Usuario | |
| Prioridad del negocio: Alta | Riesgo en desarrollo: Bajo |
| Descripción: Muestra la lista de grupos de repositorios y se marca el repositorio a eliminar. | |
| Observaciones: | |

| Tarea de Ingeniería | |
|---|---|
| Número Tarea: 13 | Número Historia de Usuario: historia de usuario 8 |
| Nombre Tarea: Programar funcionalidad eliminar grupo de repositorios. | |
| Tipo de Tarea: Desarrollo. | |
| Programador Responsable: Iván Martínez Pupo. | |
| Descripción: Funcionalidad encargada de eliminar un grupo de repositorios por su identificador. | |

**Casos de prueba para la iteración 3**

| Escenario 1. Mostrar grupos de usuarios | HU : 1 |
|---|---|
| Descripción:<br>El usuario escoge la opción de listar grupos de usuarios. | Resultados esperados:<br>Muestra el listado de grupos de usuarios. |
| Resultados obtenidos:<br>Muestra el listado de grupos de usuarios.<br>Correcto. | |
| Observaciones: | |

| Escenario 1. Listar permisos | HU : 9 |
|---|---|
| Descripción:<br>El usuario escoge la opción de listar permisos y el sistema muestra un listado de los permisos. | Resultados esperados:<br>El sistema muestra un listado de los permisos. |
| Resultados obtenidos: Correcto. | |
| Observaciones: | |

| Escenario 1.Listar un grupo de repositorios | HU : 5 |
|---|---|
| Descripción:<br>El usuario escoge la opción de Listar un grupo de usuarios y el sistema te muestra un listado de grupos de repositorios. | Resultados esperados:<br>El sistema muestra el listado de grupos de repositorios. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el grupo de usuarios fue eliminado correctamente.<br>Correcto. | |
| Observaciones: | |

| Escenario 1. Eliminar un grupo de repositorios | HU : 8 |
|---|---|
| Descripción:<br>El usuario escoge la opción de eliminar un grupo de repositorios y el sistema te muestra un listado de grupos de repositorios del cual se eliminara el seleccionado. | Resultados esperados:<br>Elimine el grupo de repositorios seleccionado y guarde los cambios.<br>Muestra un cartel informando de que se ha eliminado correctamente el grupo de repositorios. |
| Resultados obtenidos:<br>Muestra un cartel informando de que el grupo de repositorios fue eliminado correctamente.<br>Correcto. | |
| Observaciones: | |

## Conclusiones

El cumplimiento de las tareas de investigación permite obtener como resultado una herramienta capaz de administrar múltiples repositorios Git. A partir del desarrollo de las tareas de investigación se pudo llegar a las siguientes conclusiones:

- ✓ La sistematización de los antecedentes de las aplicaciones en el ámbito internacional concluyó que ninguna aplicación hasta el momento satisface las necesidades de administración de repositorios Git del proyecto Nova.
- ✓ Solo mediante el uso de tecnologías y herramientas libres, y con la metodología XP se pudo obtener en tan poco tiempo y con las actuales funcionalidades la herramienta Gitadmin.
- ✓ El empleo de Gitadmin permite a usuarios menos involucrados en la administración de repositorios Git obtener resultados satisfactorios de administración sin poseer grandes conocimientos.
- ✓ La correcta integración de la API de Gitadmin con Redmine permitirá adicionarle a este último un conjunto de funcionalidades de administración de repositorios Git que en estos momentos no posee.

**Recomendaciones**

Una vez concluido el presente trabajo investigativo y tras cumplir exitosamente los objetivos planteados, se recomienda:

- ✓ Desplegar la herramienta producto de esta investigación en el proyecto Nova para que sus servicios puedan ser de utilidad a los administradores de repositorios Git.
- ✓ Desarrollar nuevas versiones de la aplicación, donde se adicionen nuevas funcionalidades con el objetivo de que se haga más práctica y eficiente.
- ✓ Realizar estudios de nuevas tecnologías y herramientas que pudieran integrarse con la herramienta desarrollada para agregarle nuevas funcionalidades.
- ✓ Implementar un módulo usando la API desarrollada que permita la administración de múltiples repositorios Git a Redmine.
- ✓ Realizar una capacitación para el uso y modificación de la herramienta, permitiendo a sus usuarios obtener un mayor provecho de la herramienta.

**Referencia Bibliográfica:**


[1] Beck, Kent, Extreme Programming Explained, Addison-Wesley, 2000.

[2] Thomas, Dave, Programming Ruby: The Pragmatic Programmers' Guide, The Pragmatic Bookshelf Dallas, 2004.

[3] Eriksson, Hans-Erik, Business Modeling with UML: Business Patterns at Work, John Wiley & Sons, 2000.

[4] Gamma, Erich, Elements of Reusable Object-Oriented Software, Addison-Wesley, 2003

[5] Object Management Group. Unified Modeling Language Specification.2001

[6] Introducción al HTML, w3schools.com, 2010, Disponible en: http://www.w3schools.com/html/html_intro.asp , consultado: Enero 2010.

[7] Introduccion a Javascript, w3schools.com, 2010, Disponible en: http://www.w3schools.com/js/js_intro.asp, consultado: 3 Febrero 2010.

[8] Metodologías de desarrollo del software, latecladeescape.com, Disponible en: http://latecladeescape.com/w0/ingenieria-del-software/metodologias-de-desarrollo-del-software.html, consultado: Febrero 2010.

[9] Mendoza, María, ¿Qué metodología debo usar para el desarrollo de un Software?, informatízate, Disponible en: http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html, consultado: Febrero 2010.

[10] The Nuts & Bolts of Project Kenai, Kenai, Disponible en: http://kenai.com/projects/help/pages/KenaiOverview, consultado: Enero 2010.

[11] Indefero, Indenfero.com, Disponible en: http://www.indefero.net/, consultado: Enero 2010.

[12] Redmine, Redmine.com, Disponible en: http://www.redmine.org/, consultado: Enero 2010.

[13] Gitorious, Gitorious.com, Disponible en: http://gitorious.org/about, consultado: Febrero 2010.

**Bibliografía:**


Arquitectura Cliente/Servidor. [En línea] http://www.csi.map.es/csi/silice/Global71.html.

Chacon, Scott. Git, the fast versión control system. [En línea][Citado el: 10 de Febrero 2010] http://git-scm.com/about

El Guru Programador. [En línea] [Citado el: 02 de Enero de 2010.] http://www.elguruprogramador.com.ar/articulos/que-es-una-api.htm.

El Modelo Cliente/Servidor. [En línea] http://agamenon.uniandes.edu.co/~revista/articulos/cliser.html.

García, Luis. Sistema de control de versiones: Subversion. Educación. Observatorio Tecnológico. [En línea] [Citado el: 15 de Marzo de 2010] http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=548.

Historia de los Sistemas de Control de Versiones. [En línea][Citado el: 20 de Enero 2010] http://davidjguru.wordpress.com/2010/03/11/sistemas-de-control-de-versiones-iii-un-poco-de-historia/

HTML Con Clase-Indice. HTML Con Clase. [En línea] [Citado el: 2 de Febrero de 2010.] http://html.conclase.net/tutorial/html/

Introducción a los sistemas de control de versiones. [En línea] http://www.lug.fi.uba.ar/documentos/scms/index.php#prologo

León, Eduardo. Tutorial Visual Paradigm for UML. [En línea] http://www.slion2000.blogspot.com.

Linus Torvalds contento con Git. [En línea][Citado el: 6 de Marzo 2010] http://www.somoslibres.org/modules.php?name=News&file=article&sid=2732

Manual Page. [En línea]. [Citado el: 6 de Febrero 2010] http://www.kernel.org/pub/software/scm/git/docs/

Ortega, Victor Daniel. Sistemas de Control de Versiones. [En línea][Citado el: 5 de Enero 2010] http://www.wiziq.com/tutorial/14983-Sistemas-de-Control-de-Versiones

Programming languages on the internet. Web Developers Notes. [En línea] [Citado el: 02 de 12 de 2009.] http://www.webdevelopersnotes.com/basics/languages_on_the_internet.php3.

Repositorios digitales. Información bibliográfica. [En línea] [Citado el: 14 de Enero de 2010.] http://web.usal.es/~angelpoveda/web%20biologia/tutoriales/cat%C3%A1logos,%20repositorios%20y%20bibliotecas%20virtuales1/repositorios_digitales.html.

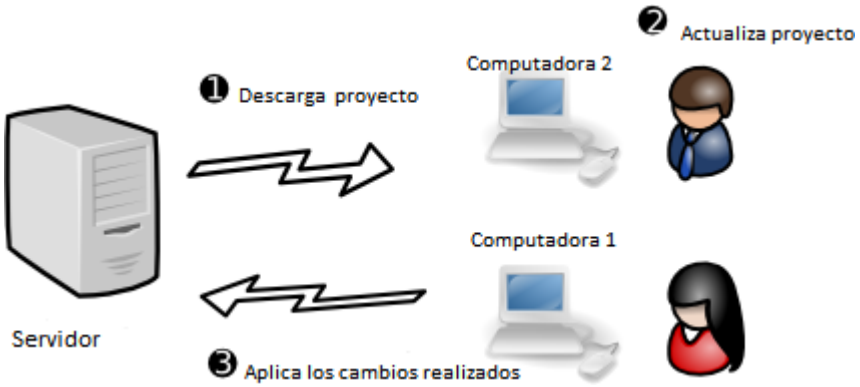
Sistemas de Control de Versiones: ¿Centralizados o distribuidos? [En línea][Citado el: 18 de Enero 2009] http://bosqueviejo.net/2009/01/12/sistemas-de-control-de-versiones-%C2%BFcentralizados-o-distribuidos/

Sistemas de Control de Versiones Libres. [En línea][Citado el: 15 de Enero 2010] http://producingoss.com/es/vc-systems.html

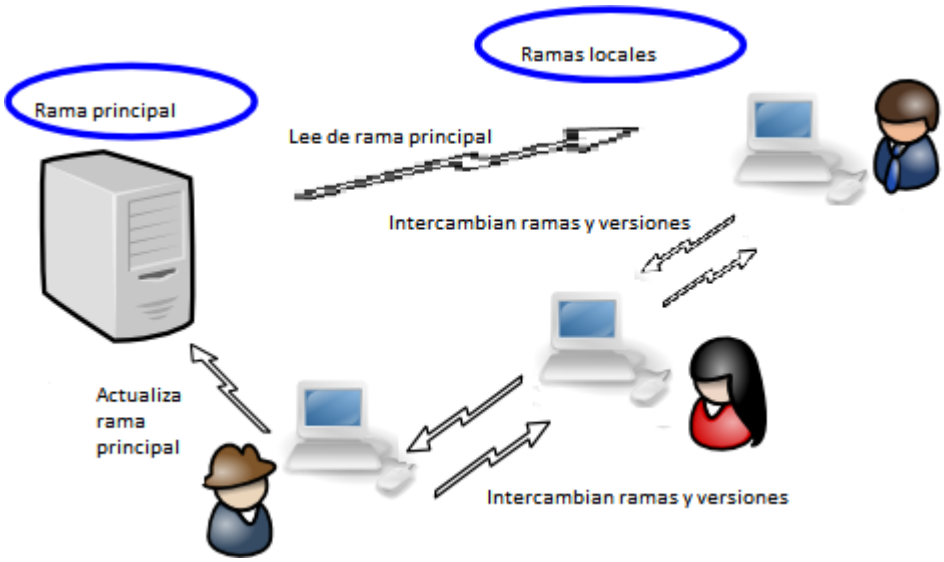
Torvalds, Linus. Git. [En línea]. [Citado el: 4 de Marzo 2010] http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918

**Anexos:**

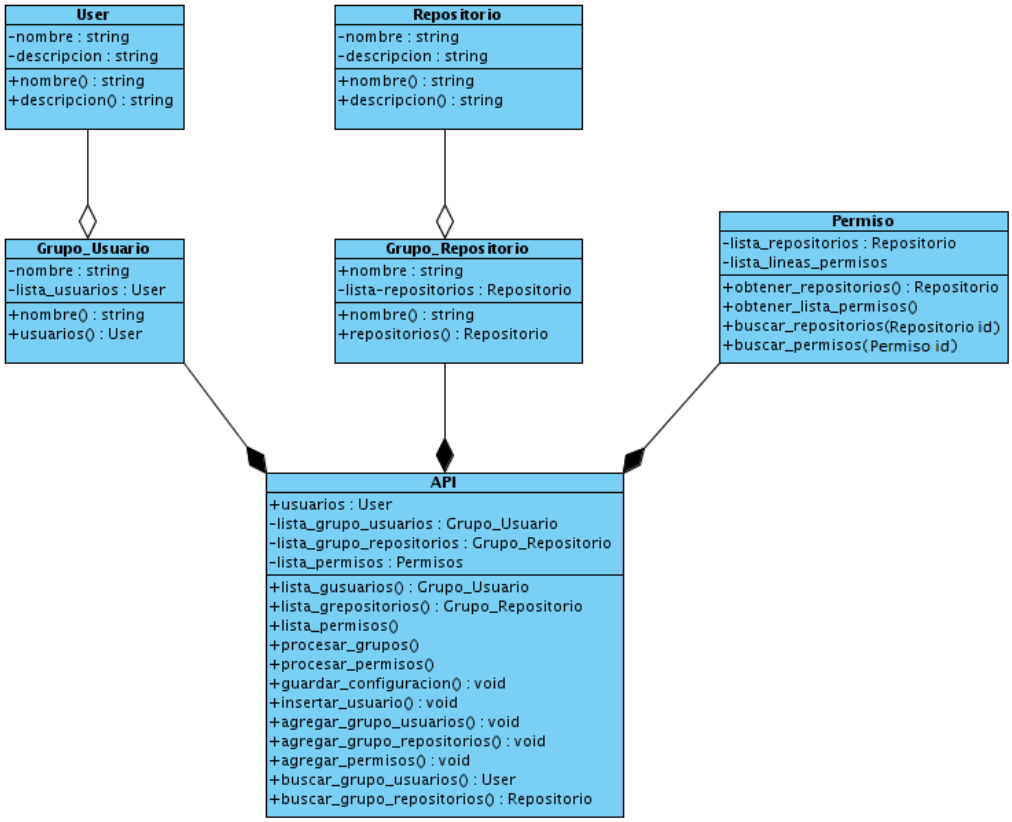
**Anexo 1: Repositorio Centralizado.**




**Anexo 2: Repositorio Distribuido.**



## Anexo 3: Diagrama de Clases.



**Glosario de Términos:**

**API** (Interfaz de Programación de Aplicaciones (sigla en inglés)): conjunto de rutinas, procedimientos, protocolos, funciones y herramientas que una determinada biblioteca pone a disposición para que sean utilizados por otro software como una capa de abstracción.
**CVS** (Sistema de Control de Versiones Concurrentes): es una herramienta que mantiene el registro de todo el trabajo y los cambios en la implementación de un proyecto informático (de software) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren en el mismo.
**HTML** (Lenguaje de Marcas de Hipertexto): es el lenguaje de marcas de texto utilizado normalmente en la World Wide Web (www). Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.
**Repositorio**: es un sitio centralizado o descentralizado que almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.
**SCV** (Sistemas de Control de Versiones): es un sistema de gestión de archivos y directorios, que mantiene la historia de los cambios y modificaciones que se han realizado sobre ellos a lo largo del tiempo.
**SVN** (Subversion): es un programa controlador de versiones empleado en la administración de archivos utilizados en el desarrollo de software o contenido.
**UML** (Lenguaje Unificado de Modelado): Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.
**Terminal**: Término derivado del inglés que significa consola.
**Metodología**: Un conjunto de métodos eficientes orientados a conseguir un objetivo propuesto. Son un conjunto de procesos que organizados dan una secuencia de pasos a seguir para obtener los hitos propuestos y finalmente el producto.
**Libre**: Software Libre o herramientas libres, se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software.
**Interfaz gráfica** (GUI): En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario, es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.
**Historias de Usuario**: Secuencias de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de las secuencias.
**GPL**: La GNU General Public License (inglés: Licencia Pública General) es una licencia

creada por la Free Software Foundation y orientada principalmente a los términos de distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es Software Libre.

**GNU/Linux**: GNU/Linux es, a simple vista, un Sistema Operativo. Es una implementación de libre distribución UNIX para computadoras personales, servidores, y estaciones de trabajo. Fue desarrollado para el i386 y ahora soporta los procesadores i486, Pentium, Pentium Pro y Pentium II, así como los clones AMD y Cyrix. También soporta máquinas basadas en SPARC, DEC Alpha, PowerPC/PowerMac, y Mac/Amiga Motorola 680x0. Como sistema operativo, GNU/Linux es muy eficiente y tiene un excelente diseño.