Universidad de las Ciencias Informáticas
Facultad 6

Título:

"Nuevo software Controlador de Versiones para el Polo de Bioinformática"

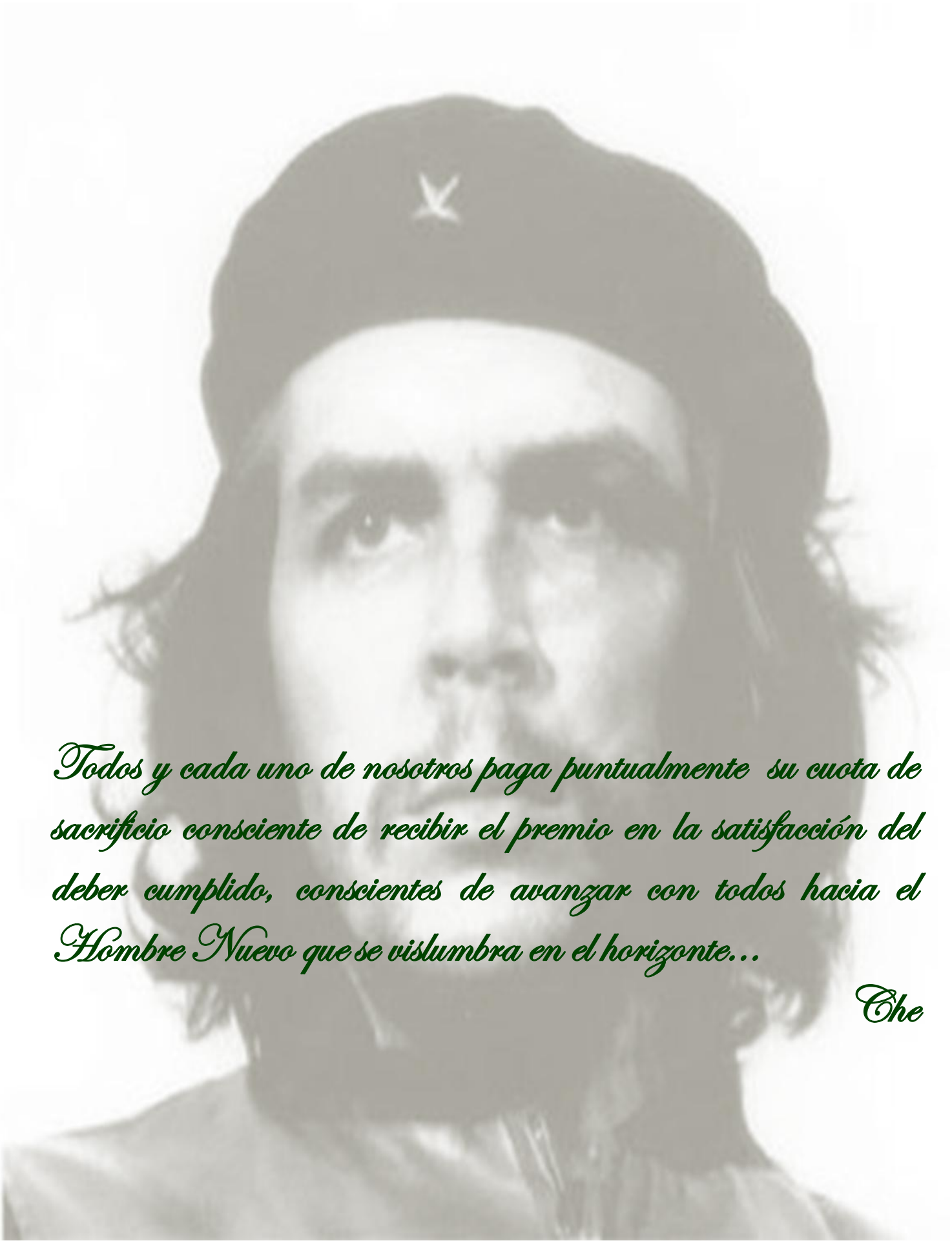
Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Lianet de León Gutiérrez

Yuneisy Medina Colina

Tutor: Msc. Noel Moreno Lemus

Ciudad de La Habana, junio de 2009



Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte…

Che

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los __ días del mes de __ del año 2009.

Lianet de León Gutiérrez	**Yuneisy Medina Colina**	**Noel Moreno Lemus**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

Datos de Contactos

Msc Noel Moreno Lemus
Licenciado en Química Nuclear
Correo electrónico: noel@uci.cu
Universidad de las Ciencias Informáticas. Ciudad de La Habana. Cuba

Agradecimientos

*A la **Revolución** y a **Fidel**, por permitirnos realizar nuestro sueño de estudiar en esta Universidad.*

*A nuestro **tutor** Noel, por su asesoramiento científico y estímulo para seguir creciendo intelectualmente, por su ayuda y sugerencias en la realización de esta investigación.*

*A nuestros **suegros** Graciela y Ernesto, por brindarnos su apoyo en todo momento.*

*A **Veylys**, gracias por brindarnos siempre tu ayuda incondicional.*

A todas aquellas personas que de una forma u otra han aportado un granito de arena en nuestra formación como futuros profesionales.

Agradecimientos

A mi mamá y mi papá, por estar siempre muy cerca y brindarme todo su amor incondicional.

A mi familia, por ayudarme a superar cada obstáculo y estar siempre junto a mí.

A mi novio, por su amor y apoyo.

A mis amigos y amigas por compartir tantas alegrías y buenos momentos.

Lianet

A mis padres, por darme siempre su apoyo y estar en cada momento de mi vida. Por ser verdaderos guías y ejemplo. Por brindarme tanto amor, cariño y comprensión en cada minuto. Por haber contribuido en la realización de este sueño y haberme formado tantos valores para ser la mujer que soy.

A mi papá, gracias por existir y ser el mejor padre del mundo.

A mi mamá, gracias por darme siempre toda su dedicación y estar a mi lado.

A mi prima Yusleidy, que estuvo siempre en cada momento de mi niñez ayudándome, nunca olvidaré los buenos momentos que pasamos juntas.

A mi tía Juana, que ya no está presente, pero sé que estaría orgullosa de verme hoy, porque sé que me quiso como una hija.

A mi novio, gracias por estar a mi lado cada instante.

A todas aquellas personas que estuvieron presentes en algún momento de mi vida e influyeron que este gran sueño se hiciera realidad.

Yuneisy

Dedicatoria

A mis abuelos Nena y Papio por darme tanto amor.

A mi mamá y mi papá por darme la vida y creer en mí, por ser guías y ejemplo.

A mi osito por hacer que cada minuto a su lado sea único.

Lianet

.

A mi papá, por ser el mejor hombre del mundo y enseñarme que en la vida todo se logra con esfuerzo y sacrificio.

A mi mamá, por quererme tanto y enseñarme que la vida es una flor que abre sus pétalos cada año, el 3 de marzo.

A mi novio, por dedicarme todo su amor y para que siga mi ejemplo.

Yuneisy

Resumen

Uno de los mayores retos a los que se enfrentan los desarrolladores de software es administrar los cambios y mantener estructurado los avances, retrocesos y modificaciones que sufre el producto mientras es implementado. En la presente investigación se realiza un estudio de los controladores de versiones, haciendo énfasis en los distribuidos. Recogiendo las principales potencialidades de cada uno de ellos, permitiendo hacer comparaciones entre los mismos para finalmente seleccionar el controlador que se implantará en el Polo de Bioinformática, dado que el controlador de versiones que se encontraba anteriormente presenta deficiencias que retrasan e interfieren el trabajo del equipo de desarrollo. Como resultado de la investigación se obtuvo que el controlador de versiones Bazaar presenta características que lo hacen superior al resto de los software estudiados. Además se implementó un prototipo funcional de aplicación con los comandos básicos del controlador para facilitar el trabajo de los desarrolladores.

Índice

Introducción

La Comunidad Primitiva dejó al hombre, como ser social, un importante legado, desde los principios de su existencia, sintió la necesidad de agruparse y trabajar en conjunto, inicialmente para defenderse, cazar, recolectar y cultivar la tierra, mejorando así sus condiciones de vida y posteriormente como vía para su desarrollo social en comunidades a fines. La ayuda mutua y el intercambio de experiencias han sido características que han trascendido hasta la actualidad.

Mediante el devenir de los años, la constante evolución de la humanidad ha traído aparejado grandes avances en la ciencia y la técnica. Con los primeros pasos en la informática, el desarrollo del software era llevado a cabo por una persona o un pequeño grupo de estas, las cuales se trazaban metas y tareas a ejecutar, alcanzando así con un bajo nivel organizacional la producción del producto deseado, esto justificado por el tamaño y la poca exigencia de las aplicaciones.

Con el auge tecnológico y el vertiginoso despegue de la informática, el desarrollo de software ha alcanzado magnitudes antes inimaginables. La aparición de roles dentro del proceso de desarrollo, para responder a diferentes funciones en el seno del mismo. El incremento del uso de las aplicaciones en todas las esferas de la vida, lo cual produce un elevado número de clientes con necesidades y con más exigencias en la calidad. La gran magnitud de los proyectos, con altos coeficientes de complejidad y sus crecientes costos. Han dado lugar a la búsqueda de alternativas para organizar el trabajo y que estos aspectos cuenten con una respuesta adecuada.

Surgen de esta forma las herramientas de colaboración para la gestión del trabajo en equipo, las cuales son sistemas que permiten acceder a ciertos servicios que facilitan a los usuarios comunicarse y trabajar conjuntamente sin importar su ubicación en el mismo lugar físico. (1)

Las herramientas colaborativas se clasifican según sus funcionalidades en distintos grupos, uno de estos es donde se ubican las herramientas para el control de versiones, las cuales garantizan la gestión de los activos de software dentro de un proceso de desarrollo, así como el historial de las modificaciones realizadas en el mismo. Según la forma de almacenamiento los controladores de versiones se pueden clasificar como Centralizados y Distribuidos.

Dentro de los controladores de versiones centralizados el de mayor aceptación en la actualidad es Subversion, dado que cuenta con características que lo hacen superior a los de su tipo que lo anteceden. Esta afirmación es avalada por su eficiencia en la creación de ramas y etiquetas, por su

atomicidad en las actualizaciones, y su mejor aprovechamiento del ancho de banda. Por estas y otras razones es usado en la mayoría de los proyectos de la Universidad de las Ciencias Informáticas y en la totalidad de los proyectos del Polo de Bioinformática.

A pesar de las ventajas antes mencionadas, un profundo análisis del mismo permitió constatar que no es el más óptimo en la elaboración de grandes proyectos, ya que presenta deficiencias que retrasan e interfieren el trabajo del equipo de desarrollo, tales como: la necesidad de pasar por el servidor central para enviar los cambios al resto de los desarrolladores, la imposibilidad de fusionar automáticamente las ramas, entre muchas otras.

Por tanto se define el siguiente **problema científico de la investigación:** la insuficiencia del controlador de versiones Subversion.

Lo que conduce a definir como **objeto de estudio:** los software controladores de versiones y como **campo de acción**: los software controladores de versiones distribuidos.

En el contexto investigativo se determina como **objetivo general:** Implantar un nuevo software distribuido para controlar la gestión de versiones en el Polo de Bioinformática.

Como **objetivos específicos:**

- Seleccionar el sistema de control de versiones distribuido a usar en los proyectos del Polo de Bioinformática.
- Elaborar un manual del uso del controlador de versiones seleccionado para el trabajo en los proyectos del Polo de Bioinformática.
- Desarrollar una aplicación para el uso del controlador de versiones seleccionado.
- Implantar el sistema de control de versiones en un laboratorio del Polo de Bioinformática.
- Validar la propuesta de sistema controlador de versiones contra un proyecto concreto.

Para guiar el proceso investigativo se determinan las siguientes **tareas científicas:**

- Realización de un estudio de la evolución y actualidad de los controladores de versiones.
- Determinación de los criterios de selección.
- Selección de los controladores de versiones más estables.

- Establecimiento de comparaciones entre estos controladores.
- Selección del controlador de versiones a utilizar.
- Estudio de los comandos del controlador de versiones seleccionado.
- Elaboración del manual.
- Levantamiento de requisitos para el desarrollo de un prototipo funcional de aplicación.
- Implementación del prototipo funcional.
- Uso del controlador de versiones en un proyecto productivo concreto.
- Aplicación de encuestas a los miembros del proyecto.

Para la ejecución de las tareas se aplicaron los siguientes **métodos científicos**:

Métodos teóricos:

Análisis y síntesis: se utilizó con el objetivo de realizar un estudio la de bibliografía consultada para profundizar y realizar valoraciones sobre los fundamentos teóricos de la evolución y actualidad de los controladores de versiones.

Inducción y deducción: posibilitó determinar las características generales de los controladores de versiones permitiendo comparar los centralizados y los distribuidos, así como plantear sus ventajas.

Métodos Empíricos:

Encuesta: Se aplicó un cuestionario de preguntas a los integrantes de los proyectos BioSyS y FreeViews para obtener información directa sobre el grado de aceptación del controlador de versiones Bazaar.
Entrevista: Se utilizó para constatar el conocimiento que poseen los integrantes del proyecto sobre las deficiencias que presenta Subversion.

El presente trabajo se desarrollará en cuatro capítulos:

Capítulo 1: Fundamentación Teórica

En este capítulo se recogerán los principales conceptos para la comprensión plena de los temas tratados en el resto del documento. Se realizará un estudio comparativo de los controladores de versiones existentes, profundizando para una posterior selección, en los distribuidos. Además se abordarán las metodologías y herramientas utilizadas.

Capítulo 2: Propuesta de implantación del controlador de versiones seleccionado

En este capítulo atendiendo a los criterios de selección planteados, se seleccionará el Controlador de Versiones óptimo para el trabajo en el Polo de Bioinformática. Se describirá además el proceso de diseño y confección del manual de uso del controlador de versiones seleccionado. Se explicará la distribución de las estaciones de trabajo en un proyecto concreto dentro del Polo de Bioinformática.

Capítulo 3: Desarrollo de un prototipo funcional de aplicación

En este capítulo se detallará la elaboración de un prototipo funcional de aplicación para el controlador de versiones seleccionado. Se realizará un levantamiento de requisitos y descripciones de los casos de uso.

Capítulo 4: Validación de la propuesta

En este capítulo se describirá el proceso de aplicación de encuestas para analizar la aceptación y los resultados obtenidos.

Capítulo 1 Fundamentación teórica

En este capítulo se recogerán los principales conceptos para la comprensión plena de los temas tratados en el resto del documento. Se realizará un estudio comparativo de los controladores de versiones existentes, profundizando para una posterior selección, en los distribuidos. Además se abordarán las metodologías y herramientas utilizadas.

1.1 Herramientas Colaborativas

La creación de software es una actividad compleja que requiere de la colaboración de grandes equipos de personas. Una de las habilidades que más se valoran en el desarrollo de software es la capacidad de trabajo en equipo, para optimizar y organizar el mismo surgen las herramientas colaborativas: sistemas que permiten acceder a ciertos servicios que facilitan a los usuarios comunicarse y trabajar conjuntamente sin importar su ubicación en el mismo lugar físico. (1)

Las herramientas colaborativas presentan varias características comunes: son multiusuario, ya que permiten que varios usuarios accedan al mismo recurso e intercambien información (2). Además son fáciles de usar y están diseñadas para el trabajo en red, ya que posibilitan la comunicación entre usuarios con fines comunes.

Existen varios tipos de herramientas para la colaboración, dentro de estas se encuentran las herramientas de Gestión Documental tal es el caso de las Wikis y los Blog que permiten al usuario crear contenidos, combinando textos, imágenes y videos. También existen herramientas de Gestión de Proyectos y Listas de Correo para difundir una información de interés a un grupo de usuarios. Otras herramientas colaborativas son las de Control de Versiones que son sistemas que gestionan las diferentes versiones por las que transita un software.

Estas herramientas permiten a un grupo de trabajo, realizar proyectos desde lugares remotos, no únicamente desde las instituciones a las cuales están vinculados, lo que genera cambios en términos de tiempo y gastos económicos. Además se puede elaborar un proyecto, manipular la información y no sólo leerla o tener acceso a ella, sino modificarla.

1.2 Software Controladores de Versiones

El control de versiones combina procedimientos y herramientas para gestionar las versiones de los elementos que se producen en el desarrollo de un software. Este facilita a los usuarios la administración de los mismos y conocer en qué momento y fase se encuentran. (3)

Para llevar a cabo el proceso de control de versiones surgen los software controladores de versiones que son herramientas que facilitan el trabajo en paralelo de grupos de usuarios. Indican la evolución de los diferentes módulos del proyecto y disponen de un control detallado de los cambios que se han realizado; funciones que son indispensables durante la vida del proyecto.

Los controladores de versiones deben proporcionar además un mecanismo de almacenamiento de los elementos que deba gestionar, posibilidad de realizar cambios sobre los elementos almacenados y registro histórico de las actividades desarrolladas. La utilización de estos sistemas es clave en el desarrollo de cualquier producto en el que interviene más de una persona.

El primer Sistema de Control de Versiones realizado fue CVS y consta de 1986 (4), desde ese entonces muchos proyectos se han efectuado y en la actualidad la utilización de estas herramientas es muy frecuente en la producción de software por grandes equipos de desarrollo, por lo que existe una gran cantidad de estos. Los mismo se pueden dividir en dos grandes grupos, los comercializados por empresas que consideran el código fuente un activo más que tienen que mantener en propiedad imposibilitando el acceso de terceros y los de código fuente abierto, desarrollados por individuos, grupos o empresas que permiten el acceso libre y la modificación del código. Entre los primeros se pueden citar: ClearCase, Perforce, Visual SourceSafe y otros. Dentro de los de licencias libres se destacan: Bazaar, Git, Subversion, CVS, entre otros. (5)

1.2.1 Ventajas del uso de Controladores de Versiones

- **Control exacto**: Saber cuál es la última versión del código, quién y cuándo la ha cargado. (6)
- **Comparación de versiones**: Ver cuáles han sido los cambios realizados. (6)
- **Regresar atrás**: Cuando el trabajo desarrollado no nos ha dado los resultados esperados, se puede retroceder a una versión anterior. (6)

- **Crear distintas ramas del proyecto**: Si llegado a un punto se hace necesario hacer dos aplicaciones con distintas funcionalidades, pero con elementos en común, se pueden separar en dos ramas. (6)
- **Concurrencia del trabajo**: Donde más de una persona puede trabajar con el mismo archivo. (6)
- **Etiquetar distintas revisiones:** Para indicar que la revisión 4283 coincide con la versión del proyecto 2.4.0, la etiquetaremos por ejemplo: 2_4_0. (6)

1.2 Criterios de Selección de un Controlador de Versiones

Para seleccionar un controlador de versiones hay que tener en cuenta una serie de características que los distinguen.

- **Mecanismo de almacenamiento:** Los controladores de versiones se dividen en dos grandes grupos según su forma de almacenamiento **Centralizados** siguen la arquitectura cliente/servidor para acceder a los datos y **Distribuidos** trabajan de forma independiente donde el flujo de información no tiene que ser a través de un servidor central.
- **Modo de funcionamiento:** Este puede estar dado en **Cliente/Servidor** donde un cliente o varios hacen peticiones al servidor donde se encuentran almacenados los datos o **Independiente** no hay un servidor que mantenga una copia del repositorio, sino que está mantenida entre los clientes que estén en uso del repositorio, mientras más desarrolladores haya conectados, mejor conectividad habrá entre todos.
- **Flujos de Trabajo:** Se crean según las necesidades del equipo de desarrollo, teniendo en cuenta el controlador de versiones seleccionado.
- **Usabilidad:** Ser fáciles de entender y utilizar.
- **Seguridad:** Presentar distintos niveles de seguridad para la integridad de los datos.
- **Soporte:** La herramienta debe tener soporte tanto por parte de los creadores como por otros desarrolladores.
- **Sistema Operativo:** Una buena herramienta de software debe ser multiplataforma. (7) (8)

1.4 Centralizado vs Distribuidos

Los Controladores de Versiones Centralizados se basan en un repositorio central único, al que todos los desarrolladores se conectan para reportar cambios. Suelen basarse en una línea de tiempo, es decir, los cambios guardan una dependencia lineal en el tiempo.

Estos controladores de versiones funcionan de modo Cliente-Servidor, por lo que se hace necesario que el servidor siempre esté encendido y con conexión permanente, además debe ser configurado, con algún sistema de autenticación y permisos.

Existen dos tipos de sistemas de control de versiones centralizados, los que usan el modelo Bloquear-Modificar-Desbloquear y los que usan el modelo Copiar-Modificar-Fusionar. El primero es el más simple y limitado, consiste en que cada vez que un usuario quiere modificar un archivo, este se bloquea y no puede ser modificado por nadie más hasta que este usuario termine de editarlo. El referido modelo, además de ser muy limitado e incómodo, rompe el concepto de cambio ya que estos están centrados en archivos y en cambios al repositorio como un todo. El segundo modelo propone lo siguiente: se hace una copia del estado actual del repositorio, se modifica y se aplica el cambio al repositorio central haciendo una fusión. (9)

Entre los controladores de versiones centralizados más conocidos tenemos a: CVS, Subversion, ClearCase y Perforce.

Por otro lado los controladores de versiones distribuidos están diseñados de manera que cada copia de trabajo es en realidad un repositorio completamente funcional. Con un sistema de control de versiones distribuido es más fácil para los desarrolladores trabajar de forma independiente sin tener que obtener permiso para acceder a un repositorio centralizado.

Además se utiliza el sistema *peer to peer,* donde los usuarios pueden comunicarse con otros y mantener sus propias ramas locales, esta sincronización tiene lugar entre los pares que deciden que partes intercambiar. Esto trae como ventajas que las operaciones normales como commits, vista de históricos y diferencias, entre otras son más rápidas porque no hay necesidad de comunicarse con un sistema central. De esta forma aumenta la robustez de la disminución de la dependencia de un único punto de falla potencial. Además cada copia es vista como un *backup* remoto proporcionando así un sistema natural de seguridad contra pérdida de datos. (10) [Anexo 1]

Con un sistema distribuido se puede trabajar desconectado, solo se necesita estar conectado para compartir cambios, de esta forma se elimina la dependencia de estar siempre en red. Actualmente el desarrollo de controladores de versiones distribuidos ha tomado un gran auge, los que gozan de mayor mérito son: Bazaar, Git y Mercurial.

Después de analizar las ventajas, desventajas y características de cada uno, vemos que el uso de los sistemas distribuidos es el más indicado, sobre todo, para proyectos con gran afluencia de desarrolladores.

1.5 Descripción de las principales características de las herramientas de control de versiones

1.5.1 Subversion

Subversion, también conocido como SVN, es un sistema de control de versiones centralizado que se ha popularizado bastante, en especial dentro de la comunidad de desarrolladores de software libre. Está preparado para funcionar en red, y se distribuye bajo una licencia libre de tipo Apache. (11)

Surge con la intención de sustituir y mejorar al controlador de versiones CVS (Concurrent Versions System) que pese a sus limitaciones, constituyó el estándar de facto de los sistemas de gestión de versiones en el ámbito del software libre. (11)

Los creadores de Subversion, CollabNet, Karl Fogel, Ben Collins-Sussman y otros desarrolladores, luego de catorce meses de trabajo, el 31 de agosto del 2001 dejan de utilizar CVS y comienzan usar SVN para la administración de su propio código fuente. (11)

El conocido software maneja ficheros y directorios a través del tiempo. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos o examinar el historial de cambios de los mismos. En este aspecto, muchas personas piensan en los sistemas de versiones como en una especie de “máquina del tiempo”. (12)

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varios usuarios puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar

todas las modificaciones. Si el trabajo se encuentra bajo el control de versiones, no hay razón para temer que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único, si se ha realizado un cambio incorrecto a los datos, simplemente se puede deshacer ese cambio. (12)

Características de Subversion

- **Versionado de directorios:** Implementa un sistema de ficheros versionado "virtual" que sigue los cambios sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones. (12)
- **Verdadero historial de versiones:** Se puede añadir, borrar, copiar y renombrar ficheros y directorios. Cada fichero nuevo añadido comienza con un historial nuevo, limpio y completamente suyo. (12)
- **Envíos atómicos:** Una colección cualquiera de modificaciones, o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito. (12)
- **Versionado de metadatos:** Cada fichero y directorio tiene un conjunto de propiedades claves y sus valores asociado a él. Se puede crear y almacenar cualquier par arbitrario de clave/valor que se desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros. (12)
- **Elección de las capas de red:** Tiene una noción abstracta del acceso al repositorio, facilitando a las personas implementar nuevos mecanismos de red. Puede conectarse al servidor HTTP Apache como un módulo de extensión. Esto proporciona a Subversion una gran ventaja en estabilidad e interoperabilidad y acceso instantáneo a las características existentes que ofrece este servidor: autenticación, autorización, compresión de la conexión, etcétera. También tiene disponible un servidor de Subversion independiente y más ligero. Este servidor tiene un protocolo propio, el cual puede ser encaminado fácilmente a través de un túnel SSH. (12)
- **Manipulación consistente de datos:** Expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto (legibles para humanos) y ficheros binarios (ilegibles para humanos). Ambos tipos de ficheros son almacenados e igualmente comprimidos en el repositorio y las diferencias son transmitidas en ambas direcciones a través de la red. (12)

- **Ramificación y etiquetado eficientes:** El coste de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Subversion crea ramas y etiquetas simplemente copiando el proyecto, usando un mecanismo similar al enlace duro. De este modo estas operaciones toman solamente una cantidad de tiempo pequeña y constante. (12)
- **Hackability:** No tiene un equipaje histórico; está implementado como una colección de bibliotecas compartidas en C con APIs bien definidas. Esto hace a Subversion extremadamente fácil de mantener y reutilizable por otras aplicaciones y lenguajes. (12)

1.5.2 Mercurial

Mercurial, es una herramienta de control de versiones distribuido y multiplataforma. Está implementado principalmente haciendo uso del lenguaje de programación Python, pero incluye una implementación binaria de *diff* escrita en C. Fue escrito originalmente para funcionar sobre Linux. Ha sido adaptado para Windows, Mac OS X y otros sistemas tipo Unix. Mercurial es, sobre todo, un programa para la línea de mandatos. Todas sus operaciones se invocan como opciones dadas a su programa motor. Su sigla es hg, ya que hace referencia al símbolo químico del mercurio. (13)

Las principales metas de desarrollo de Mercurial incluyen un gran rendimiento y escalabilidad; desarrollo completamente distribuido, sin necesidad de un servidor; gestión robusta de archivos tanto de texto como binario y capacidades avanzadas de ramificación e integración, todo ello manteniendo sencillez conceptual. Además incluye una interfaz web integrada. (13)

El creador y desarrollador principal de Mercurial es Matt Mackall. El código fuente se encuentra disponible bajo los términos de la licencia GNU GPL versión 2, lo que clasifica a Mercurial como software libre. (13)

Características de Mercurial

- **Transaccional**: Cada cambio sobre el repositorio se ejecuta atómicamente para todos los elementos implicados en el mismo; sean archivos, directorios y/o metadatos. (13)
- **Es un sistema distribuido**: El histórico de los archivos gestionados no se encuentra en un único servidor centralizado al que los desarrolladores suben sus cambios. Cuando un desarrollador descarga los archivos de un proyecto de Mercurial, éste descarga además una copia completa del histórico de cambios. De esta manera el desarrollador es capaz de

reproducir el estado de la aplicación en cualquier momento de su historia sin necesidad de conectarse a un repositorio centralizado. (13)

- **Lenguaje de programación:** Desarrollado en Python y algo de C. (13)
- **Funciona en modo Push y Pull:** Esto le da flexibilidad a las políticas de versionado a aplicar. (13)

1.5.3 Git

Git es un sistema de control de versiones diseñado para manejar proyectos muy grandes; con velocidad, eficiencia y confiabilidad en el mantenimiento de versiones de aplicaciones con una enorme cantidad de archivos de código fuente, pero igual de apropiado para repositorios pequeños. Es especialmente popular en la comunidad Open Source, sirviendo como plataforma de desarrollo para proyectos como el Kernel Linux, Ruby on Rails, WINE o X.org. (14)

Está dentro de la categoría de herramientas de manejo de código fuente distribuido, similar a Mercurial o Bazaar. Cada directorio de trabajo es un repositorio completo con historial y capacidades totales de *tracking* de revisiones, independiente de acceso de red o un servidor central. Aún así, es extremadamente rápido y eficiente con el espacio. (14)

Es un proyecto Open Source cubierto por la GNU General con licencia pública v2. Originalmente escrito por Linus Torvalds y mantenido por Junio Hamano C. (14)

El diseño se basó sobre BitKeeper y en Monotone. En principio, Git se pensó como un motor de bajo nivel que otros pudieran usar para frentes como Cogito o StGIT. Sin embargo, se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. (14)

Características de Git

Entre las características más relevantes se encuentran:

- **Fuerte incidencia en la no linealidad de los cambios**: Rapidez en la gestión de ramificaciones y mezclado de diferentes versiones. (14)
- **Gestión distribuida**: Los cambios se importan como ramificaciones y pueden ser mezcladas en la manera en que lo hace una ramificación del almacenamiento local. (14)
- **Protocolos:** Los almacenes de información pueden publicarse por HTTP, FTP, SSH, rsync o mediante un protocolo nativo, aparte de ser posible emular CVS. (14)

- **Gestión eficiente:** Rapidez de gestión de diferencias entre archivos y otras mejoras de optimización de velocidad de ejecución. (14)
- **Autenticación criptográfica de historial**: Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio. (14)
- Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja en base a cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos. (14)
- **Renombrados de Ficheros**: Los renombrados se trabajan en base a similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre en base a supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles coincidencias de ficheros diferentes en un único nombre. (14)
- **Realmacenamiento periódico en paquetes (ficheros):** Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura, si el reempaquetado (en base a diferencias) no ocurre cada cierto tiempo. (14)

1.5.4 Bazaar

Bazaar desarrollado por Canonical Ldt. (Patrocinador comercial de Ubuntu), está escrito en Python y publicado bajo la licencia GNU/GPL. Es un sistema de control de versiones acorde a la colaboración y cooperación en el núcleo del software libre. Facilita el uso de control de versiones distribuido a través de todas las plataformas (Windows, Linux y Max OS X). (15)

Permite guardar progresivamente los cambios que se vayan realizando sobre un conjunto de archivos de texto (habitualmente código fuente), recuperar versiones anteriores, mostrar diferencias, integrar el trabajo de diversos programadores, se centra en la facilidad de uso y puede ser utilizado por uno o más desarrolladores. (15)

Entre las muchas ventajas que presenta, se encuentra su adaptación a los flujos de trabajo de una forma mucho más flexible para el desarrollador, desde el típico esquema cliente-servidor hasta la descentralización de los repositorios. (15)

No es necesario dedicar un servidor con Bazaar instalado, es suficiente un acceso FTP a un servidor web. Se pueden alojar ramas de Bazaar en casi cualquier servidor donde haya acceso de lectura y escritura sobre SFTP, FTP, HTTP, o sobre SSH con el *smart server* de Bazaar. Se dispone de un

servidor inteligente para los que necesitan mejorar el rendimiento o la seguridad, pero no es necesario en la mayoría de los casos. (15)

Bazar tiene un sistema de plugins en Python API, siendo flexible de integrar a las herramientas y proyectos y a la vez, fácil de extender o integrar con la infraestructura existente. La integración vía XML es soportada con un plugins. Permite el seguimiento de la configuración de archivos aún cuando el equipo de trabajo está bien distante. (15)

El diseño de Bazaar intenta ser lo más directo posible en cuanto a la interoperabilidad con proyectos que usan un sistema de control de versiones distinto. Su meta es que sea posible que se use Bazaar para participar en cualquier proyecto. Así que puede representar las operaciones de cualquier otro sistema, y hay plugins que permiten leer el log de versiones para un proyecto en Bazaar desde Subversion, Git, CVS entre otros. Por ejemplo, se puede mantener el *trunk* del desarrollo de un proyecto en Subversion y correr "*import*" constantemente de ese t*runk* en Bazaar de donde los desarrolladores pueden hacer *branch y merge*. (15)

Es un sistema seguro, está respaldado por la comunidad de código abierto y patrocinado por Canonical, una de las compañías de más rápido crecimiento en torno a las empresas de código abierto. El proceso de desarrollo sigue las mejores prácticas, con la revisión de código de todos los cambios fundamentales y de la comunidad. Tiene una enorme suite de pruebas (más de 10.000 pruebas), que asegura que las nuevas características puedan ser rápidamente añadidas sin romper las existentes. (15)

Características de Bazaar

- **Fusión Inteligente:** La fusión (*Merge*) reconoce bien los cambios idénticos, minimiza los falsos conflictos y es capaz de fusionar solo lo que falta. (15)
- **Fiable:** Desarrollado bajo un extenso repositorio de prueba, verifica y comprueba las ramas en cualquier momento. (15)
- **Fácil de Usar:** Tiene aproximadamente una docena de comandos básicos, pero con solo cinco de ellos se puede obtener buenos resultados. Presenta una gran ayuda para todos los comando, incluso para los más esotéricos. (15)
- **Portátil:** Escrito en Python, trabaja y está formalmente probado en una variedad de plataforma con fácil instalación. (15)

- **Intuitivo:** Gestiona ramas con los comandos de Sistemas Operativos, permite una fácil migración desde CVS y Subversion. (15)
- **Exclusivamente poderoso:** Realiza seguimiento a los cambios de archivos y directorios. Soporta nombres de archivos Unicode, ramas (Branches) removibles y uncommit. (15)
- **Gran documentación y soporte:** Tiene un sitio web bien organizado. Facilidades para encontrar la información y leer la documentación. Tiene un buen canal de IRC y las listas de correos están activas. (15)
- **Buena Integración:** Es parte de una docena de sistemas operativos, configurable a través de las herramientas 3rdPartyTools. Tiene plugins para integrarse a otros controladores de versiones incluyendo Subversion. (15)
- **Ajustable:** Establece valores predeterminados para los usuarios y las ramas. (15)
- **Flexible:** Tiene aproximadamente una docena de plugins disponibles. Multiplataforma incluyendo GNU / Linux, FreeBSD, Mac OS X, Windows. Soporta múltiples modelos de desarrollo. (15)
- **Totalmente software libre:** No depende de software propietario, puede utilizarse para desarrollar cualquier software, es totalmente software libre. (15)

1.6 Flujos de Trabajo con Bazaar

Bazaar nos permite trabajar de una forma más flexible, adaptándose al flujo de trabajo que queramos utilizar. Este epígrafe nos brinda la posibilidad de conocer detalladamente los flujos de trabajo de Bazaar.

Estrategia Solo

Esta forma de trabajo es la más simple que presentan los Sistemas de control de Versiones (CVS), generalmente se utiliza con sistemas descentralizados, debido a que en los centralizados se requieren permisos y un sistema accesorio de control de acceso. Es efectiva en proyectos pequeños o proyectos donde trabaja una sola persona. (15) [Anexo 2]

Estrategia en Pareja

El flujo de trabajo en pareja es usado de desarrollos pequeños a medianos, ideal para la programación en pares, requiere de una herramienta de control de versiones descentralizada, esta forma es efectiva

para proyectos donde haya dos desarrolladores trabajando en conjunto y necesiten compartir los cambios que realicen con una combinación inteligente de los datos. Cada línea de texto en cada fichero se atribuye a un cambio concreto, incluyendo quién lo cambió, cuándo y por qué. (15) [Anexo 3]

Estrategia Descentralizada con línea principal compartida

Este modo de trabajo es ideal para proyectos con un numeroso equipo de desarrollo, donde cada trabajador tiene su propia rama o ramas de trabajo independiente; todos tienen autorización para poder enviar cambios a la rama principal. Los desarrolladores pueden combinar entre sí los cambios en sus ramas personales cuando están trabajando en algo común, hacen su trabajo en su rama personal y lo combinan con la principal sólo cuando está totalmente lista. Esta forma ofrece una organización sencilla de trabajo. (15) [Anexo 4]

Estrategia Descentralizada con supervisor humano

En la estrategia descentralizada con supervisor humano, como en la forma anterior, cada desarrollador tiene su propia rama o ramas de trabajo, más un acceso de sólo lectura a la rama principal. Un desarrollador (el supervisor) tiene permiso de *commits* en la rama principal. Cuando un desarrollador quiere que se combine su trabajo, le pide al supervisor que lo haga. El supervisor revisa el código y lo combina con la rama principal si cumple con los estándares necesarios. La principal ventaja de este modo de trabajo es que el código siempre se revisa antes de pasar a la línea principal. (15) [Anexo 5]

Estrategia Descentralizada con supervisor automático

En este modo de trabajo, cada desarrollador tiene su propia rama o ramas, más acceso de sólo lectura a la rama principal. Un software supervisor (por ejemplo *Patch Queue Manager* (PQM)) tiene permisos de escritura en la línea principal. Cuando un desarrollador quiere que su trabajo se combine en la rama principal, pide a otra persona que lo revise. Una vez que se ha revisado, o bien el autor, o bien el revisor, pide al supervisor que lo combine, según la política de trabajo del equipo. El supervisor hace un *merge*, compila y ejecuta los *test*. Si está correcto, combina el código en la rama principal. Como alternativa, el paso de revisión se puede omitir y el autor puede enviar el cambio al supervisor automático sin revisión. Esto es especialmente apropiado cuando se usan prácticas como la programación en parejas, que efectúan revisión de código sobre la marcha según se hace, en vez de realizarlo como un paso separado. Su principal ventaja radica en que el código, que siempre se prueba antes de que entre en la rama principal, por lo que la integridad de la misma es alta, permite también un mejor escalado cuando el equipo crece. (15) [Anexo 6]

Bazaar permite trabajar también de forma centralizada, presenta para esto, dos flujos de trabajo, pero no se va a hacer énfasis sobre estas estrategias, por las desventajas que representa trabajar con ellas.

1.7 Metodologías de desarrollo de software

Se define como metodología al conjunto de estrategias, procedimientos, métodos o actividades intencionadas, organizadas, secuenciadas e integradas, que permitan el logro de aprendizajes significativos y de calidad. (16)

A su vez, una metodología para el desarrollo del proceso de software es el conjunto de actividades necesarias para transformar los requisitos de los usuarios en un sistema software.

En realidad no existe una metodología estándar, sino que las características de cada proyecto, las de su equipo de desarrollo, recursos disponibles y tiempo para su elaboración exigen la flexibilidad del proceso, adaptándose el mismo al entorno y teniendo como objetivo alcanzar la máxima calidad en lo que se produce.

Dentro de las más conocidas en la actualidad se encuentran OpenUp, Proceso Unificado de Desarrollo (RUP) y Programación Extrema (XP).

1.7.1 OpenUp

OpenUp es un Proceso de Desarrollo de Software creado por la fundación Eclipse. La mayoría de los elementos del mismo están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Es un proceso interactivo de desarrollo de software simplificado, completo y extensible. Es utilizado por equipos de desarrollo que valoran los beneficios de la colaboración y de los involucrados con el resultado del proyecto, por encima de formalidades innecesarias. Este está caracterizado por cuatro principios básicos interrelacionados:

- Colaboración para unificar intereses y compartir conocimientos.
- Equilibrio de prioridades competentes a maximizar el valor de los involucrados con el resultado del proyecto.
- Enfoque en la articulación de la arquitectura.

- Desarrollo continuo para obtener realimentación y realizar las mejoras respectivas.

Establece un equilibrio entre las necesidades de los involucrados con los resultados del proyecto y los costos técnicos. Además desarrolla un ciclo de vida iterativo que mitiga el riesgo a tiempo y ofrece demostrar resultados en curso al cliente del proyecto. (17) (18)

Esta ha sido la metodología seleccionada por el Polo de Bioinformática de la UCI para el desarrollo de sus proyectos, además, por presentar las características de disminuir los riesgos y permitir la realización de software en un corto periodo de tiempo será la que se utilizará para la ejecución de la propuesta.

1.7.2 Metodología XP

Se encuentra dentro del grupo de metodologías ágiles. Intenta reducir la complejidad del software por medio del trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción.

XP pretende minimizar el riesgo de fallo del proceso, por medio de la disposición permanente de un representante competente del cliente a disposición del equipo de desarrollo. El mismo debe estar apto para contestar de forma rápida y concreta cualquier duda que pueda surgir por parte del equipo, de manera que la toma de decisiones no se retrase.

Está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, donde la comunicación sea más factible que en grupos de desarrollo grandes. Presenta como características fundamentales la comunicación, simplicidad y realimentación. (19)

1.8 Herramientas para el desarrollo de software

1.8.1 Lenguaje de programación

Python

Durante las últimas dos décadas se han desarrollados varios lenguajes de programación, que ofrecen gran flexibilidad, portabilidad y robustez a las aplicaciones, ejemplo de esto es el lenguaje interpretado Python creado por Guido van Rossum en el año 1991.

Se compara habitualmente con Perl, Java y Ruby y en la actualidad se desarrolla como un proyecto de código abierto. Python permite dividir el programa en módulos reutilizables y viene con una gran colección de estos que se pueden utilizar como base. También hay módulos incluidos que proporcionan llamadas al sistema, sockets y hasta interfaces a GUI (interfaz gráfica con el usuario), entre otros.

Se utiliza como lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo de éste. El principal objetivo que persigue es la facilidad, tanto de lectura, como de diseño. (20)

Este lenguaje tiene otras características importantes que determinaron que se seleccionara para la implementación de nuestra aplicación, tales como:

- **Orientado a Objetos:** Python trabaja con sus datos como objetos. Soporta las características propias del paradigma orientado a objetos: herencia múltiple y polimorfismo.
- **Simple:** Posee una curva de aprendizaje muy rápida. Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estos.
- **Elementos del Lenguajes:** Python fue diseñado para ser leído con facilidad. Entre otras cosas se utilizan palabras en inglés donde otros lenguajes utilizarían símbolos.
- **Extensión fácil:** Nuevos módulos se pueden escribir fácilmente en C o C++, puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesiten una interfaz programable.
- **Programación multiparadigma:** Permite varios estilos como son Programación Orientada a Objetos, Programación Estructural y Programación Funcional. (21)

1.8.2 Entorno de Desarrollo Integrado

NetBeans

NetBeans es un Entorno Integrado de Desarrollo libre y gratuito. Es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java

pero puede servir para cualquier otro lenguaje de programación. Es una base modular y extensible, usada como una estructura de integración para crear grandes aplicaciones de escritorio.

Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases para interactuar con las APIs de NetBeans y un archivo especial que lo identifica como módulo. Debido a que estos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. (22)

Eclipse

Eclipse es un entorno de desarrollo integrado, multiplataforma y de código abierto. Éste emplea módulos para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido. El mecanismo de módulos permite a Eclipse extenderse usando otros lenguajes de programación como son: C, C++ y Python. Además de proveer soporte para Java.

En cuanto a las aplicaciones clientes, Eclipse, proporciona al programador un conjunto de *Frameworks* para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Lo que permite obtener interfaces personalizables y profesionales. Presenta varias características:

- Resaltado de sintaxis.
- Compilación en tiempo real.
- Asistentes para creación de proyectos, clases, test, etc.
- Refactorización. (23)

Luego de realizar un estudio de los IDEs NetBeans y Eclipse, se decidió que el Entorno de Desarrollo Integrado NetBeans no se debía utilizar en la propuesta, ya que en su versión 6.5 no permite diseñar aún interfaces gráficas para el lenguaje Python. Para la programación con este lenguaje se le agregó a Eclipse el plugins *PyDev*, que se integra a este IDE y permite programar en el referido lenguaje. Es una de las variantes más utilizadas, ya que cuenta con varias características tales como: un buen manejo de errores en la gramática, resaltado y análisis de sintaxis. Además presenta depurador gráfico, refactorización y la posibilidad de realizar interfaces gráficas para el lenguaje Python.

1.8.3 Herramienta de Modelado

Visual Paradigm

Es una herramienta CASE que utiliza UML como lenguaje de modelado. Soporta el ciclo de vida completo del desarrollo de software y agiliza la construcción de aplicaciones de calidad a un menor costo. Permite dibujar todos los tipos de diagramas de clases, realizar ingeniería inversa, generar documentación y código desde diagramas.
Visual Paradigm brinda:

- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Disponibilidad de múltiples versiones para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.

Se integra con herramientas como: Eclipse, NetBeans, MS Visio y otras. (24)

Es la herramienta seleccionada para realizar el modelado de la propuesta, ya que permite la realización de los diagramas necesarios y es multiplataforma.

Capítulo 2 Propuesta de implantación del controlador de versiones seleccionado

En este capítulo, atendiendo a los criterios de selección planteados, se seleccionará el Controlador de Versiones más óptimo para el trabajo en el Polo de Bioinformática. Se describirá además el proceso de diseño y confección del manual de uso del controlador de versiones seleccionado, para ello se especificarán las instrucciones de instalación y las principales funcionalidades a utilizar. Se explicará la distribución de las estaciones de trabajo en un proyecto concreto dentro del Polo de Bioinformática.

2.1 Propuesta del Controlador de Versiones

A continuación se realiza una comparación entre los cuatro controladores de versiones de licencia libre más utilizados en la actualidad; atendiendo a los criterios de selección planteados por la investigación, con el fin de detectar potencialidades y desventajas en cada unos de ellos. Obteniendo como resultado final de la confrontación, el software controlador de versiones que se utilizará en el Polo de Bioinformática.

CRITERIOS	Controladores de Versiones			
	Subversion	**Mercurial**	**Git**	**Bazaar**
Mecanismo de almacenamiento	Centralizado	Distribuido	Distribuido	Distribuido
Modo de funcionamiento	Cliente/Servidor	Independiente	Independiente	Independiente
Flujos de trabajos	Predeterminado	Predeterminado	Predeterminado	Personalizado
Usabilidad	Alta	Alta	Media	Alta
Seguridad	Permisos de acceso (lectura, escritura)	Permisos de acceso (lectura, escritura)	Validación criptográfica	Validación criptográfica (firmas digitales)
Soporte	CollabNet, Inc.	Comunidad de software libre	GNU General	Canonical Ltd.
Sistema Operativo	GNU/Linux, Windows, Mac OS	GNU/Linux, Windows, Mac OS	GNU/Linux, Windows, Mac OS	GNU/Linux, Windows, Mac OS

Atendiendo a las potencialidades presentadas por el controlador de Versiones Bazaar, fundamentalmente que su mecanismo de almacenamiento es distribuido, sus flujos de trabajo son personalizables y su adaptabilidad a las características necesarias para el desarrollo en equipo en el Polo de Bioinformática, se determina que es el indicado para desempeñar dicha función.

2.2 Elaboración del Manual de Uso del software seleccionado.

Bazaar es un sistema de control de versiones multiplataforma, este epígrafe pretende dar una visión de cómo utilizar Bazaar en Ubuntu, ya que es el sistema operativo que utiliza el Polo de Bioinformática.

2.2.1 Instrucciones de instalación

Para instalar Bazaar en Ubuntu se debe realizar el siguiente paso: Abrir un terminal y teclear: **sudo apt-get install bzr** como se muestra en la figura #1.

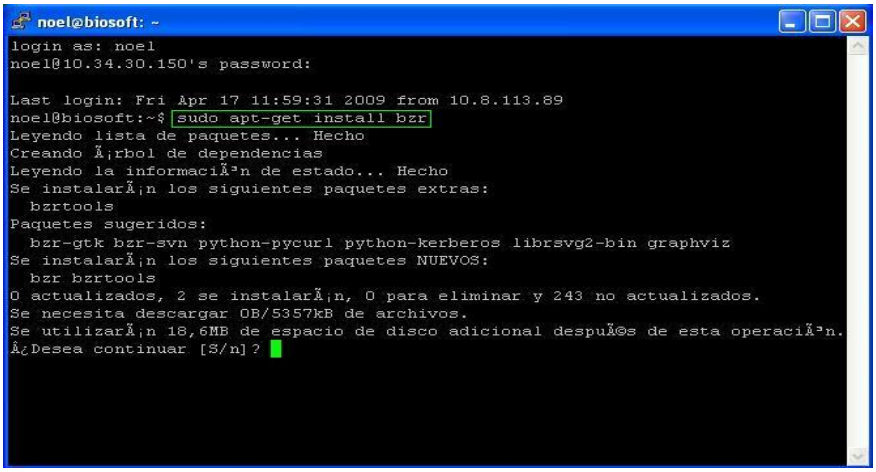


Figura #1: En la figura se muestra el terminal abierto donde se debe teclear el comando para la instalación. Se debe presionar luego la tecla **S** para que prosiga la instalación. Una vez terminado ya se dispone de Bazaar instalado. (15)

2.2.2 Configuración e Identificación

Antes de comenzar el trabajo con Bazaar, es conveniente identificarse. De ese modo el trabajo será identificado correctamente en los log de revisión.
Se debe teclear el nombre y dirección de correo como se muestra en la figura #2.

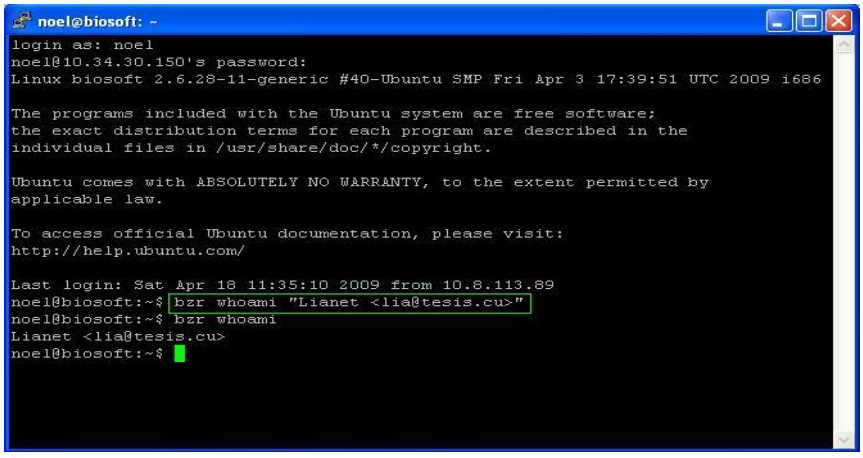


Figura #2: Muestra el comando que se debe teclear para lograr una correcta identificación.

Bazaar creará o modificará ahora un archivo de configuración, incluyendo su nombre y dirección de email.

2.2.3 Poner archivos bajo control de versiones

Se deben crear directorios y archivos para trabajar con Bazaar usando los comandos **mkdir** para crear un nuevo directorio o carpeta, **cd** para abrir un directorio o carpeta y **touch** para crear un archivo, como se muestra en la figura #3. (15)

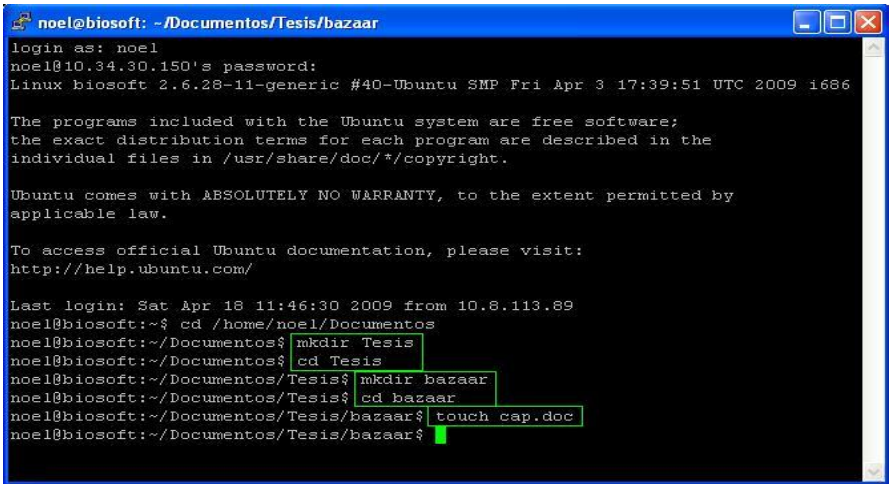


Figura #3: Muestra los comandos a utilizar para crear directorios y archivos.

Para que Bazaar se inicialice en el directorio se debe escribir **bzr init** y de esta forma se creará una rama donde se guardará archivos e históricos de revisiones.

2.2.4 Principales comandos para trabajar con Bazaar

En el siguiente epígrafe, se describirá el uso de los principales comandos de Bazaar utilizados en los proyectos del Polo de Bioinformática.

Adicionar archivos

El siguiente paso, es decirle a Bazaar qué archivos desea gestionar. Ejecutando **bzr add** agregará recursivamente todos los elementos dentro del proyecto. (15)

```
bzr add
```

Subir archivos a la rama

A continuación se debe agregar los archivos a la rama. Además de un mensaje para explicar el por qué se hace el *commits*.

```
$ bzr commit -m "Importación inicial"
```

Bazaar guarda su rama y todos sus commits dentro del directorio con el que está trabajando, debe aparecer el subdirectorio **.bzr** para comprobar que se ha hecho correctamente la operación. Cada vez que se añada o haga cambios en un fichero se debe hacer **bzr commit –m "Un comentario"** para que queden guardados los cambios en la rama. (15)

Haciendo cambios en los ficheros

Para hacer cambios en un fichero primeramente se escoge el archivo que se desea modificar en el editor indicado, si se desea comprobar la operación que se ha realizado teclee el comando **bzr diff.**

Comprobando los cambios:

```
 bzr diff
=== modified file 'prueba.txt'
--- prueba.txt   2009-04-18 12:56:14 +0000
+++ prueba.txt   2009-04-18 12:46:22 +0000
@@ -0,0 +1,1 @@
+@@--##
```

Log de revisiones

Bazaar guarda el historial de su proyecto, esto permite tener un registro de las distintas versiones de cada fichero. Si se desea ver el histórico de una rama solo se debe teclear el comando **bzr log**. (15)

```
$ bzr log
------------------------------------------------------------
revno: 3
committer: Lianet <lia@tesis.cu>
branch nick: bazaar
timestamp: sat 2009-04-18 13:01:17 +0400
message:
  Txt modificado
------------------------------------------------------------
revno: 2
committer: Lianet <lia@tesis.cu>
branch nick: bazaar
timestamp: sat 2009-04-18 12:56:58 +0400
message:
  Txt creado
------------------------------------------------------------
revno: 1
committer: Lianet <lia@tesis.cu>
branch nick: bazaar
timestamp: sat 2009-04-18 12:37:00 +0400
message:
  importacion inicial
```

Volver a una revisión anterior

Bazaar también permite volver a una versión antigua del proyecto con el comando **bzr revert –r** más el número de la revisión a la que se desea volver.

```
$ bzr revert –r1
```

Si se desea volver a la versión actual, solo debe teclear:

```
$ bzr revert
```

Si se está trabajando con la Estrategia Solo, con las instrucciones explicadas hasta el momento, se puede realizar el control de versiones de un proyecto. (15)

2.2.5 Trabajo con ramas en Bazaar

Al trabajar con las estrategias propuestas por Bazaar, donde sea necesario el intercambio de información, se tendrá que realizar publicaciones y copias de los directorios que se encuentran bajo control de versiones.

Copias de ramas

Para realizar copias de las ramas de otros desarrolladores o de la rama principal, se deberá poner el comando **bzr branch** y la dirección de la rama que se desee copiar.

```
bzr branch sftp://Lia@10.34.30.150/Documento/Tesis
Adding ssh-rsa host key for 10.34.30.150
Opened sftp connection (server version 3
Branched 3 revision(s).
```

Bazaar descargará todos los archivos e históricos de revisiones. De esta forma se dispondrá de una copia de la rama, donde se puedan enviar cambios con o sin una conexión de red. (15)

Publicación en la rama principal

Si se ha realizado modificaciones en la copia local y esta representa una versión más actualizada que la que se encuentra en la rama principal con el comando **bzr push,** se pueden subir los cambios a la rama principal.

```
bzr push sftp://Lia@10.34.30.150/Documento/Tesis
```

Actualización de la rama local

En caso que en la rama principal se hayan efectuado modificaciones y no se cuenta con la última versión de estas, se deberá poner el comando **bzr pull** y de esta forma se actualizará la copia de dicha rama. (15)

```
bzr pull sftp://Lia@10.34.30.150/Documento/Tesis
```

Integración de las ramas

Mientras se efectúan cambios en la rama local, es probable que otros desarrolladores también sigan generando modificaciones en la rama principal, por tanto, los comandos anteriormente explicados fallarán y se creará un conflicto entre los ficheros.

Se deberá integrar los cambios con el comando **bzr merge** y Bazaar mezclará automáticamente los archivos. Si se han añadido o borrado líneas en la rama principal que en la copia no se han tocado, automáticamente se añadirán o borrarán.

```
$ bzr merge
Merging from saved parent location
All changes applied successfully.
```

Pero si por el contrario, se han efectuado cambios en la misma línea en ambas ramas, se creará un conflicto que Bazaar no podrá resolver de forma automática. (15)

Resolver conflictos

Bazaar dejará en el fichero la línea conflictiva repetida, una vez con la versión local y otra con la versión del repositorio original. Se debe editar el fichero y decidir cuál de las dos líneas es la correcta. Una vez terminado, se debe indicar a Bazaar que ya se han resuelto los conflictos utilizando el comando **bzr resolve**.

```
$ bzr resolve
```

Con los comandos explicados hasta el momento, se puede realizar un perfecto control de versiones con Bazaar sobre cualquier directorio o proyecto. (15)

Bazaar posee una serie de comandos descritos anteriormente, útiles en cualquier estrategia de trabajo. Dos de estas estrategias poseen, además, la ventaja de tener un revisor que verifica la integridad de los datos, que posteriormente se enviarán a la rama principal garantizando la seguridad, estabilidad y robustez de la misma. La Estrategia con Supervisor Automático consta de un software para realizar esta operación, llamado *Patch Queue Manager* (PQM), mientras que en la Estrategia con Supervisor Humano, es un desarrollador del proyecto el que se encarga de realizar dicha función.

2.3 Protocolos de transferencia de datos usados por Bazaar

Bazaar es muy flexible en este sentido, es posible que los usuarios prefieran intercambiar información a través del protocolo SFTP. Un protocolo de seguridad incorporado en la mayoría de servidores SSH, pero también puede realizar esta operación con el apoyo de otros protocolos como se muestran en la siguiente tabla. (15)

Prefijo	Descripción
file://	Acceso mediante el mismo tipo de sistema de ficheros (por defecto)
Sftp: //	Acceso mediante SFTP
Bzr: //	Un acceso rápido mediante el servidor inteligente de Bazar
Ftp: //	Acceso mediante FTP pasivo
http://	Acceso de sólo lectura a las ramas exportadas por un servidor web

2.4 Organizar un proyecto con Bazaar

Realizar el control de versiones a un proyecto con Bazaar depende de numerosos factores, entre los que se pueden citar:

- Rol del usuario
- Flujo de trabajo empleado
- Tamaño del proyecto

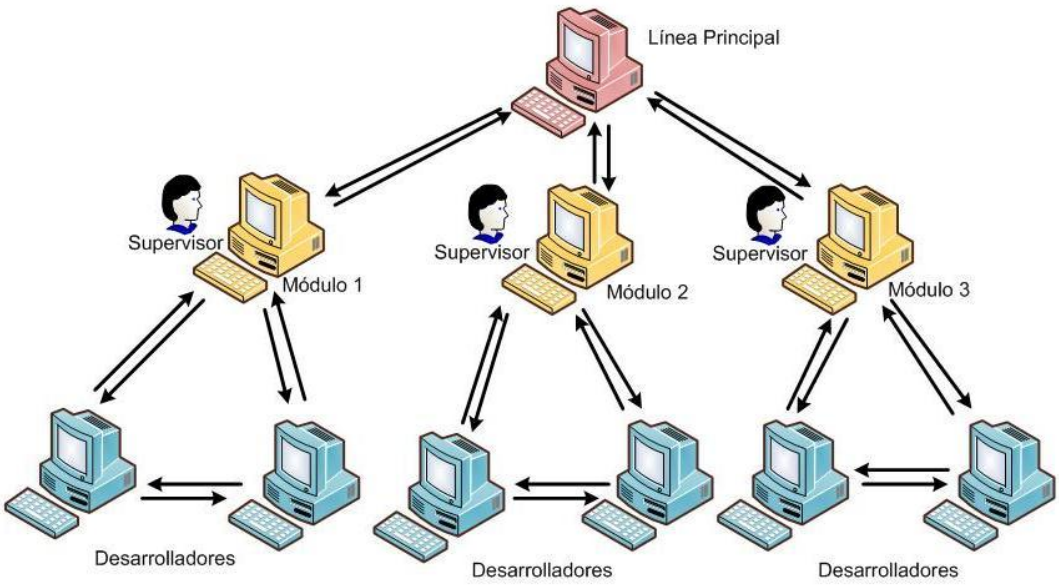
Bazaar propone varias formas de organizar un proyecto, algunas muy comunes como los árboles independientes y trabajo con ramas. Otras presentan un nivel más avanzado, como es el caso de los diseños de repositorios compartidos. Éste es eficiente en proyecto con un gran número de desarrolladores donde exista intercambio de información entre los mismos.

Bazaar dispone de un esquema que propone una distribución por niveles, primeramente se encuentra el repositorio principal del proyecto, que contendrá la versión integrada de las líneas principales de desarrollo. Estas últimas poseen información detallada y revisada que proviene de las ramas, que a su vez, se nutren de las ramas de trabajo de los desarrolladores. (15)

Proyecto/	# Repositorio principal.
+- Tronco/	# Línea principal de desarrollo del proyecto.
+- Ramas/	# Contienen los directorio.
+- Ramas locales/	# Ramas de trabajo de los desarrolladores.

2.5 Implantación de Bazaar en el proyecto BioSyS del Polo de Bioinformática.

El controlador de versiones Bazaar se implantará en el proyecto BioSyS del Polo de Bioinformática, según el esquema explicado en el epígrafe anterior, que es el que más se asemeja a las características del proyecto, este quedaría organizado de la siguiente forma:



Se debe crear una línea principal de la cual debe nutrirse todo el proyecto. Además de la creación de diferentes módulos, que a su vez, le proporcionan información a las ramas de desarrollo.

A medida que los desarrolladores vayan avanzando en la solución de la propuesta, Bazaar permite que exista un intercambio de información entre las ramas de desarrollo. De igual manera se actualicen los

módulos con los cambios realizados. Dichos módulos son los encargados de revisar la información, garantizando la fiabilidad y robustez de la misma, para luego integrarla en la línea principal.

A lo largo de la investigación se ha podido constatar varias potencialidades que presenta el controlador de versiones Bazaar. Dentro de estas, se destaca su adaptabilidad a la forma de trabajo del equipo de desarrollo, permitiendo escoger el flujo de trabajo que más se adecua a sus necesidades. Otra de las ventajas que presenta el controlador es que no depende de software propietario, puede utilizarse para desarrollar cualquier software, es totalmente libre. Además se centra en la facilidad de su uso, ya que posee una docena de comandos básicos respaldados de una extensa ayuda.

A pesar de las potencialidades antes mencionadas, Bazaar presenta algunas desventajas, como es el caso del desconocimiento de este software por una parte de la comunidad de desarrollo, siendo esto solucionado en cierta medida en el Polo de Bioinformática con la investigación realizada. Otra de las deficiencias que presenta, es que no existen aún, buenas aplicaciones visuales que permitan el trabajo con este controlador de versiones, como lo son: TortoiseSVN o RapidSVN en el caso de Subversion. Solo se ha encontrado una aplicación desarrollada en Python, denominada Olive que aún no se encuentra en su versión 1.0.

Con el objetivo de resolver esta dificultad se ha decidido hacer un levantamiento de requisitos de la posible aplicación a implementar, así como, la descripción de los casos de uso asociados a los mismos, para comprobar cuáles de estas funcionalidades ya están implementadas en Olive e implementar las que aún no lo estén o no funcionen correctamente.

Capítulo 3 Desarrollo de un prototipo funcional de aplicación.

En este capítulo se detallará la elaboración de un prototipo funcional de aplicación para el controlador de versiones seleccionado. Se describirán los requisitos funcionales y no funcionales, además de los casos de uso.

3.1 Requisitos Funcionales

Los requerimientos funcionales especifican acciones que el sistema debe ser capaz de realizar, sin tomar en consideración ningún tipo de restricción física, de manera que especifiquen el comportamiento de entrada y salida del sistema. Se pueden definir los siguientes requerimientos funcionales: (25)

RF1. Crear un nuevo directorio.

RF2. Crear un nuevo archivo.

RF3. Adicionar un directorio.

RF4. Adicionar archivo.

RF5. Subir archivos a la rama.

RF6. Mostrar los cambios realizados.

RF7. Mostrar el historial de cambios.

RF8. Volver a una versión anterior.

RF9. Copiar una rama.

RF10. Publicar en la rama principal.

RF11. Actualizar la rama local.

RF12. Integrar las ramas.

RF13. Resolver conflictos.

3.2 Requisitos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el sistema debe tener y además son características que hacen al producto atractivo, usable, rápido y confiable en muchos casos, garantizando el completo éxito del sistema, ya que están vinculado en gran medida con los requisitos funcionales. (25)

Software:

- Debe estar Instalado cualquier distribución de GNU/Linux como sistema Operativo en todas las máquinas.

Hardware:

Debe contar con procesadores:

- Pentium IV de 256 MB de memoria RAM o más.
- 20 MB de espacio en el disco duro como mínimo.
- Tarjeta de red.

Apariencia o interfaz externa:

- Diseño sencillo, permitiendo una visibilidad clara y entendible de lo que el sistema requiere.

Soporte:

- Se realizarán distintas pruebas a la aplicación una vez concluida, para comprobar su funcionalidad.

Usabilidad:

- Realizar manual de usuario.
- Las opciones deben aparecer en español.
- La aplicación debe ser usada por los integrantes del Polo de Bioinformática de la facultad 6.

Confiabilidad:

- Los fallos deben ser mínimos.
- Garantía de un tratamiento adecuado de las excepciones y validaciones de las entradas del cliente.

3.3 Descripción de los actores del sistema

Actores del sistema

Los actores de un sistema pueden ser otros sistemas o hardware externo que se relacionan o interactúan con dicho sistema, no necesariamente tiene que ser una persona. Cada actor juega un rol determinado al interactuar con el sistema y diferentes usuarios pueden asumir el mismo rol de un actor. Luego de definir el concepto de actores del sistema se puede establecer el o los actores del sistema en cuestión. (25)

	Descripción
Desarrollador	Es el encargado de realizar todas las operaciones que contiene la aplicación.

3.4 Descripción de los Casos de Uso del Sistema

Nombre del Caso de Uso	Crear directorio
Actores	Desarrollador
Propósito	Crear un nuevo directorio
Resumen	El caso de uso de inicia cuando el desarrollador crea un nuevo directorio.
Referencias	**RF1**
Precondiciones	Debe estar Bazaar instalado.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita crear un directorio.	El sistema crea una carpeta en la dirección especificada por el usuario.
Curso alterno de los eventos	

Acción del Actor	**Respuesta del Sistema**
	EL sistema debe lanzar un error si existe un directorio con el mismo nombre en la dirección especificada por el desarrollador.
Prioridad	Crítica

Nombre del Caso de Uso	Crear archivo
Actores	Desarrollador
Propósito	Crear un nuevo Archivo
Resumen	El caso de uso de inicia cuando el desarrollador crea un nuevo archivo.
Referencias	**RF2**
Precondiciones	Debe estar Bazaar instalado.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita crear un archivo.	El sistema crea un archivo en la dirección especificada por el usuario.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	El sistema debe lanzar un error si existe un archivo con el mismo nombre en la dirección especificada por el desarrollador.
Prioridad	Crítica

Nombre del Caso de Uso	Adicionar directorio
Actores	Desarrollador
Propósito	Adicionar un directorio
Resumen	El caso de uso de inicia cuando el desarrollador adiciona un directorio para ser gestionado por el controlador de versiones.
Referencias	**RF3**
Precondiciones	Debe estar el Bazaar instalado. Debe existir un directorio que contenga los archivos que se van a poner bajo control de revisiones.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita adicionar el directorio.	El sistema agregará recursivamente todos los elementos dentro del directorio
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	El sistema debe lanzar un error si el directorio ha sido adicionado anteriormente.
Prioridad	Crítica

Nombre del Caso de Uso	Adicionar archivo
Actores	Desarrollador
Propósito	Adicionar un archivo

Resumen	El caso de uso de inicia cuando el desarrollador adiciona un archivo para ser gestionado por el controlador de versiones.
Referencias	**RF4**
Precondiciones	Debe estar Bazaar instalado. Debe existir un directorio que contenga los archivos que se van a poner bajo control de revisiones.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita adicionar el archivo	El sistema agregará recursivamente todos los archivos que se encuentran dentro del directorio
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	El sistema debe lanzar un error si el archivo ha sido adicionado anteriormente.
Prioridad	Crítica

Nombre del Caso de Uso	Subir archivos a la rama
Actores	Desarrollador
Propósito	Subir archivos a la rama
Resumen	El caso de uso se inicia cuando el desarrollador hace un commit de los archivos que desea subir a rama,

	para tener una versión del proyecto guardada en el registro histórico.
Referencias	**RF5**
Precondiciones	Debe estar Bazaar instalado. Debe existir el directorio que se desea subir a la rama.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita hacer un *commit* de los archivos que desea subir a la rama y escribe un mensaje para explicar por qué se hace el *commit.*	El sistema guarda en el directorio donde está trabajando el *commit* hecho por el usuario.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	El sistema muestra un mensaje indicando que no se pudo realizar la operación.
Prioridad	Crítica

Nombre del Caso de Uso	Mostrar cambios
Actores	Desarrollador
Propósito	Ver los cambios realizados
Resumen	El caso de uso de inicia cuando el desarrollador desea ver los cambios que se han realizado en el archivo especificado.
Referencias	**RF6**

Precondiciones	Debe estar el Bazaar instalado. Debe existir un directorio que contenga los archivos que se van a poner bajo control de revisiones.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita ver los cambios que se han hecho en el archivo especificado.	El sistema busca en el registro histórico los cambios que se han hecho en el archivo especificado. El sistema muestra los cambios en una ventana, indicando la fecha y el nombre del archivo que fue modificado.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	Si no se ha realizado ningún cambio el sistema muestra un mensaje.
Prioridad	Crítica

Nombre del Caso de Uso	Mostrar historial
Actores	Desarrollador
Propósito	Ver el historial de cambios
Resumen	El caso de uso de inicia cuando el desarrollador desea ver los cambios que se han realizados en sus archivos.
Referencias	**RF7**
Precondiciones	Debe estar el Bazaar instalado.

	Debe existir un directorio que contenga los archivos que se van a poner bajo control de revisiones.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita ver su registro histórico haciendo un log.	El sistema busca en el registro histórico los archivos que están almacenados. El sistema muestra en una ventana todos los archivos que tiene en la rama indicando de cada uno el número de revisión, los datos que quién hizo la operación, la fecha y el nombre del archivo.
Prioridad	Crítica

Nombre del Caso de Uso	Volver versiones anteriores
Actores	Desarrollador
Propósito	Volver a una revisión anterior
Resumen	El caso de uso de inicia cuando el Desarrollador desea volver a una versión antigua del proyecto.
Referencias	**RF8**
Precondiciones	Debe estar el Bazaar instalado. Debe haber más de una revisión del proyecto almacenada en el registro histórico.
Poscondiciones	Aplicación lista para hacer el control

	de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
1. El desarrollador solicita volver a la versión antigua del proyecto que ha especificado. 3. El desarrollador abre el archivo en el editor indicado para comprobar la operación.	2. El sistema busca en el registro histórico la versión indicada por el desarrollador.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	En caso que no exista una versión anterior el sistema muestra un mensaje.
Prioridad	Crítica

Nombre del Caso de Uso	Copiar rama
Actores	Desarrollador
Propósito	Copiar una rama
Resumen	El caso de uso de inicia cuando el desarrollador desea hacer una copia de su rama.
Referencias	**RF9**
Precondiciones	Debe estar el Bazaar instalado.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**

El desarrollador solicita realizar una copia de su rama local.	El sistema muestra una ventana donde se debe especificar la dirección hacia donde se desea copiar la rama
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	Si la dirección es incorrecta el sistema muestra un mensaje.
Prioridad	Crítica

Nombre del Caso de Uso	Publicar rama
Actores	Desarrollador
Propósito	Publicar una rama
Resumen	El caso de uso de inicia cuando el desarrollador desea hacer una publicación su rama en la rama principal.
Referencias	**RF10**
Precondiciones	Debe estar el Bazaar instalado.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita hacer una publicación en la rama principal.	El sistema publica en la rama principal la información que tiene el desarrollador en su rama.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	En caso de que no se pueda realizar

	la operación se muestra un mensaje.
Prioridad	Crítica

Nombre del Caso de Uso	Actualizar rama local
Actores	Desarrollador
Propósito	Actualizar la rama local
Resumen	El caso de uso de inicia cuando el desarrollador desea actualizar su rama.
Referencias	**RF11**
Precondiciones	Debe estar el Bazaar instalado.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita actualizar su rama.	El sistema actualiza en la rama local las modificaciones realizadas en la rama principal.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	Si no existen nuevas modificaciones en la rama principal el sistema muestra un mensaje.
Prioridad	Crítica

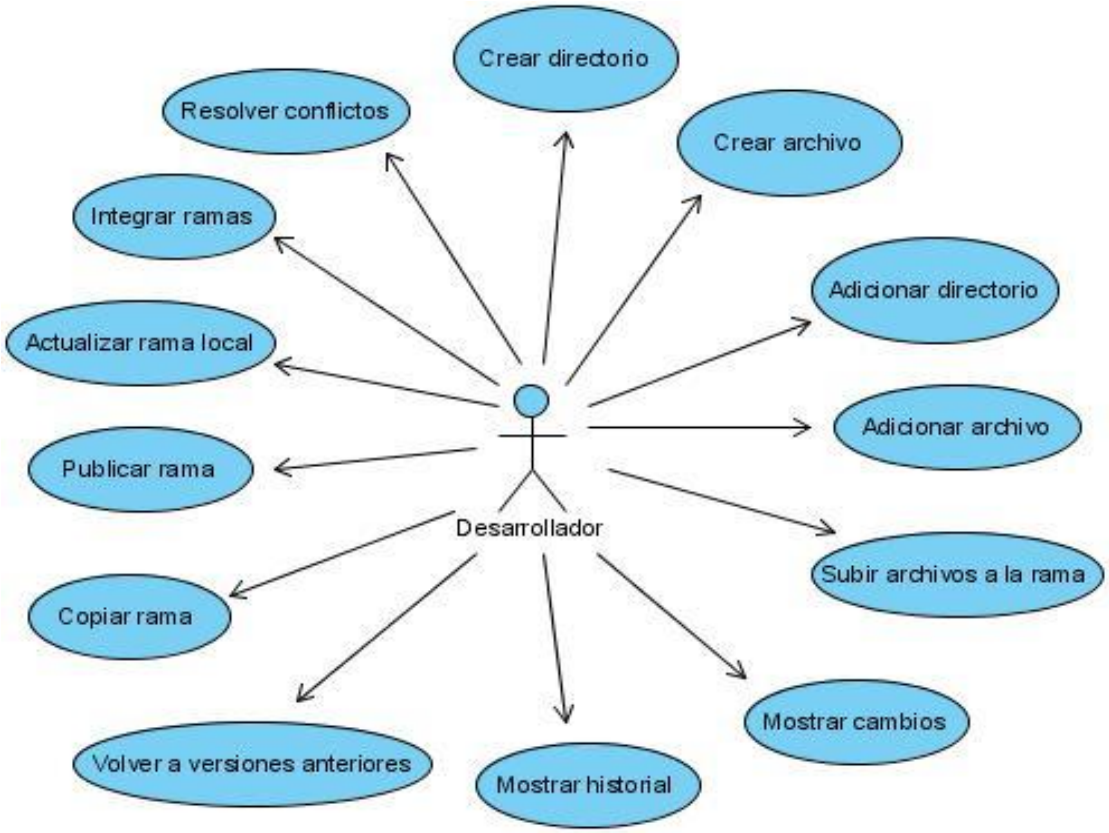
Nombre del Caso de Uso	Integrar ramas
Actores	Desarrollador

Propósito	Integración de las ramas (Locales y principal).
Resumen	El caso de uso de inicia cuando el desarrollador desea integrar su rama con la rama principal.
Referencias	**RF12**
Precondiciones	Debe estar el Bazaar instalado.
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita integrar su rama con la rama principal.	El sistema integra ambas ramas si no se han modificado las mismas líneas en sus archivos.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	El sistema debe mostrar un mensaje indicando que hay conflicto entre las ramas.
Prioridad	Crítica

Nombre del Caso de Uso	Resolver conflictos
Actores	Desarrollador
Propósito	Resolver
Resumen	El caso de uso de inicia cuando el desarrollador desea resolver los conflictos entre las ramas.
Referencias	**RF13**
Precondiciones	Debe estar el Bazaar instalado.

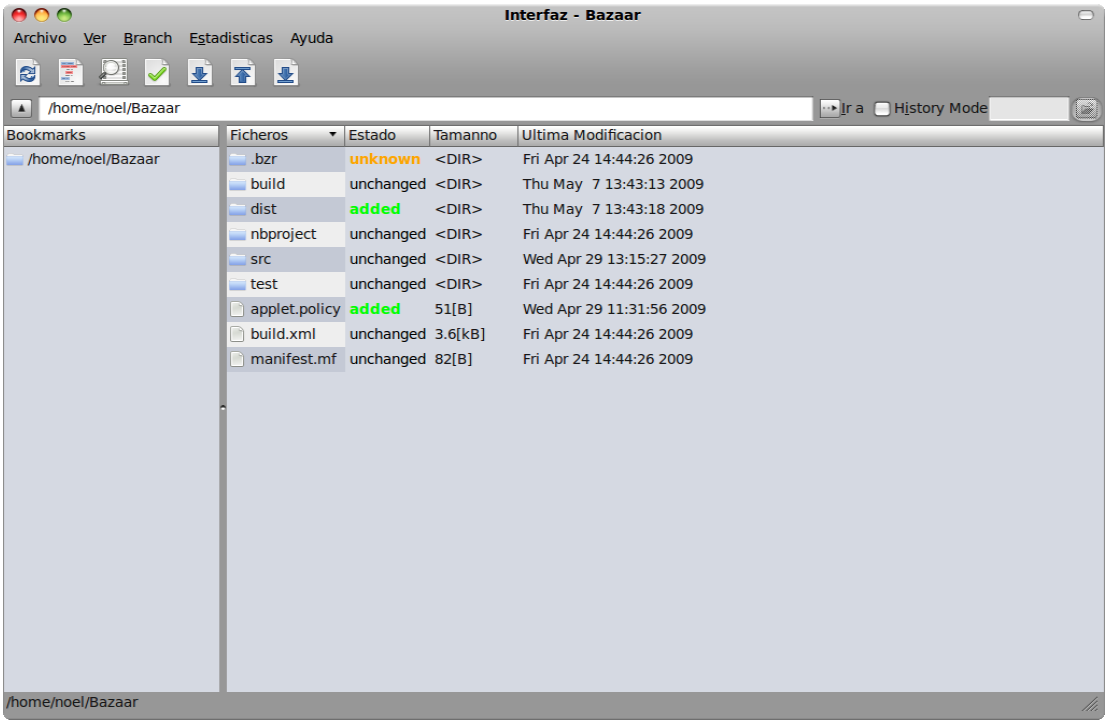
Poscondiciones	Aplicación lista para hacer el control de revisiones.
Curso Normal de los eventos	
Acción de Actor	**Respuesta del Sistema**
El desarrollador solicita resolver conflictos.	El sistema integra las ramas nuevamente una vez resuelto el conflicto.
Curso alterno de los eventos	
Acción del Actor	**Respuesta del Sistema**
	El sistema muestra un mensaje si los conflictos aún nos se han resuelto.
Prioridad	Crítica

3.5 Diagrama de Casos de Uso del Sistema

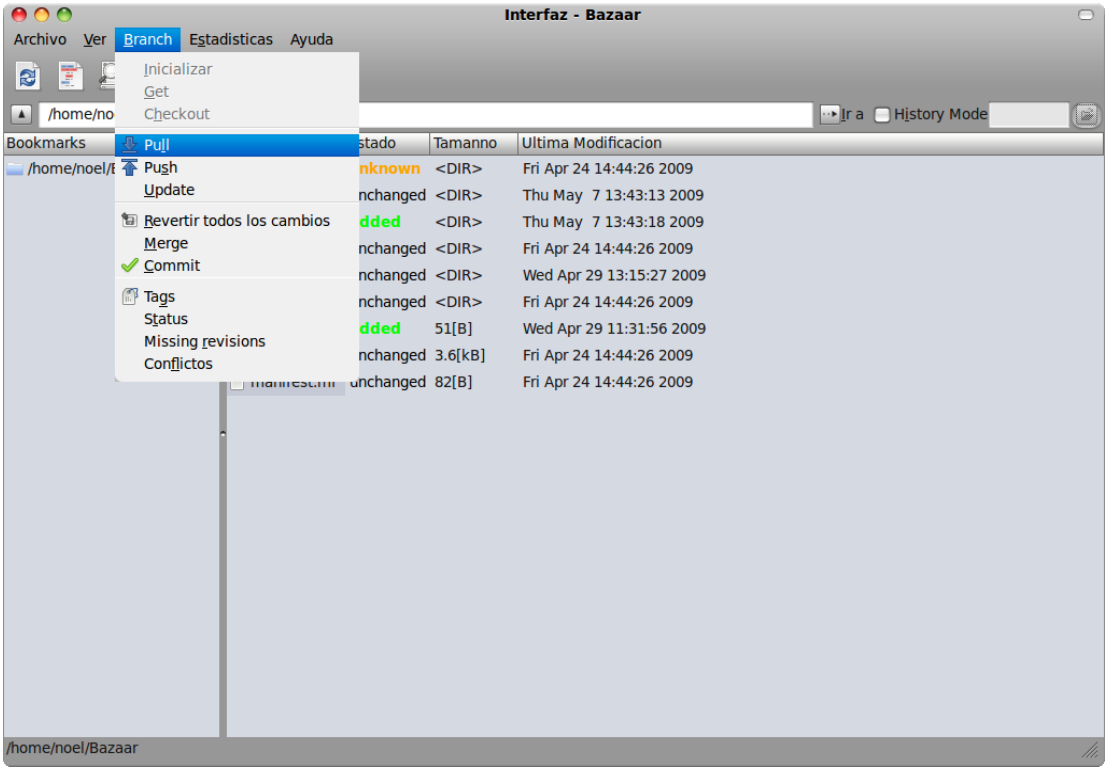


3.6 Interfaz gráfica del prototipo funcional de aplicación

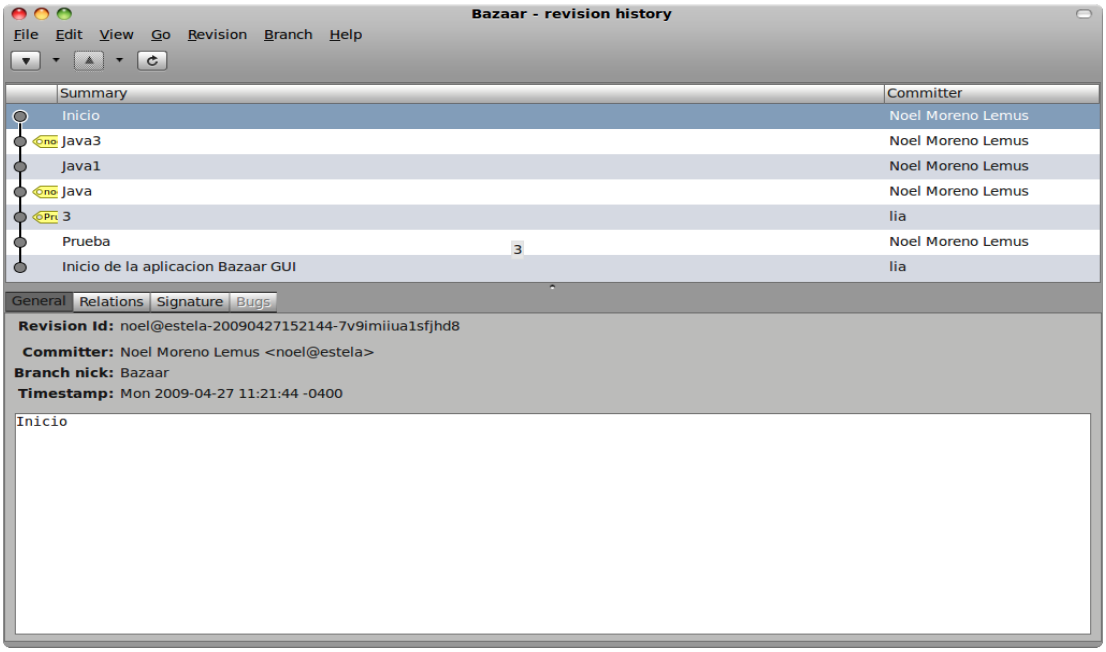
Al culminar el estudio realizado a la aplicación Olive se determinó que existían algunas funcionalidades que debían ser mejorada como es el caso de: *Copiar rama, Publicar rama* e *Integrar rama*, ya que no permitían la conexión por el protocolo sftp. Otro de los problemas que presentaba la aplicación era que la funcionalidad *Resolver conflictos* no estaba implementada, por lo que solo se podía realizar en modo consola. Además hubo que traducir toda la interfaz al español, debido a que se encontraba en idioma inglés. A continuación se muestran algunas fotos tomadas del prototipo funcional de aplicación ya implementado.



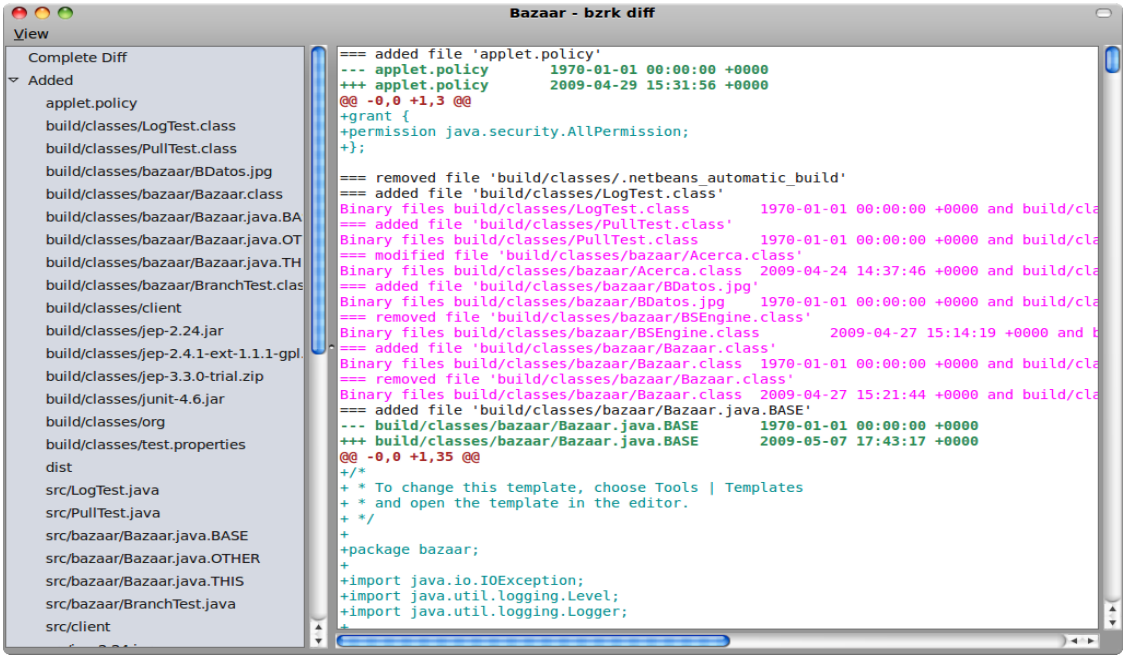
Interfaz gráfica del prototipo funcional de aplicación para Bazaar



Trabajo con ramas (Anteriormente presentaba problemas de conexión).



Revisión del historial (Muestra las versiones por las que ha transitado la información).



Muestra los cambios realizados a un documento (Diferencias).

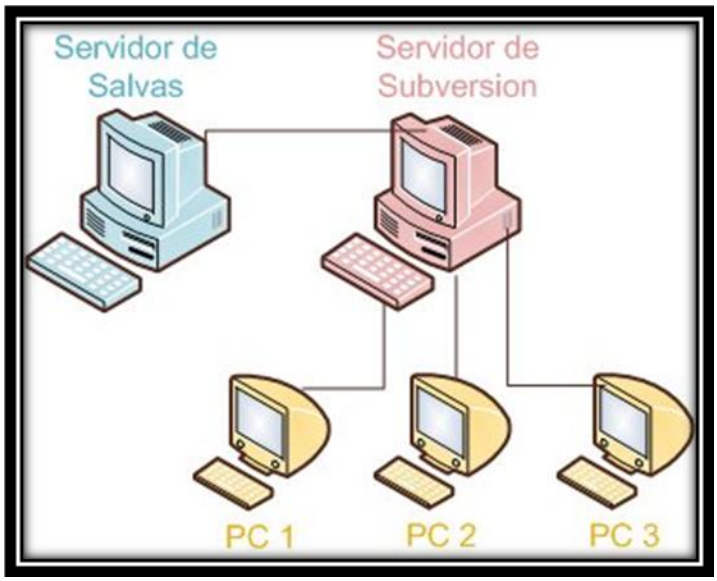
.

Capítulo 4 Validación de la propuesta

En este capítulo se describirá cómo se llevó a cabo la confección de una encuesta para comprobar el nivel de aceptación del software implantado, esto permitirá que la propuesta quede validada.

4.1 Análisis preliminar del estado de la gestión de la configuración en el proyecto BioSyS

Se realizó un análisis del estado de la gestión de la configuración en el proyecto BioSyS para saber cómo se realizaba el control de versiones con Subversion. Se contaba con una distribución siguiendo una arquitectura Cliente-Servidor. El proyecto se encontraba estructurado con un servidor Subversion donde se almacena la información y el código fuente de toda la implementación del mismo. Todas las PC del laboratorio tienen acceso a dicho servidor y tienen instaladas las herramientas Eclipse y Netbeans. La siguiente figura ilustra dicha distribución:



Esta distribución presenta como principal desventaja la dependencia total de la red, trayendo como consecuencia el retraso del equipo de desarrollo, si falla la conexión ftp, para actualizar la versión del repositorio.

Mediante la entrevista realizada a integrantes de BioSyS, se constataron limitaciones que presenta Subversion dentro del proyecto, entre las que se pueden citar: la imposibilidad de intercambiar información entre los desarrolladores, teniendo que acudir constantemente al servidor central, el

incompleto manejo de cambios de nombres de los archivos, los cuales maneja como la suma de una operación de copia y una de borrado, además no resuelve el problema de aplicar repetidamente parches entre ramas y no facilita el llevar la cuenta de qué cambios se han trasladado.

4.2 Análisis del estado actual de la gestión de configuración en el proyecto BioSyS

Luego de la implantación del software controlador de versiones Bazaar en el proyecto BioSyS, se realizó un estudio del estado de la configuración para constatar como se mantenía el control de versiones con el nuevo software y verificar si se habían eliminado las limitaciones que presentaba Subversion.

Durante el estudio se analizó la nueva estructura del proyecto para realizar el control de versiones. Se implantó la estrategia *Descentralizada con supervisor humano.* La cual consta de una línea principal que nutre a todo el proyecto, además de módulos que son los encargados de suministrarle la información a los desarrolladores y revisarla antes de ser integrada a la rama principal.

Esta nueva estructura que presenta el proyecto, permite hacer un control de versiones que facilita el trabajo de los desarrolladores, admitiendo el intercambio de información entre los usuarios que desean compartir sus ramas. Proporciona además, una mayor rapidez en las operaciones básicas, debido a que no hay necesidad de pasar por un servidor central y brinda un sistema de seguridad contra pérdida de datos, haciendo que cada copia sea vista como un backup remoto.

4.3 Resultado del estudio

Luego de haber hecho un análisis al estado de la configuración del proyecto BioSyS antes y después de haber implantado la nueva propuesta para el control de versiones, se obtuvo como resultado:

- El controlador de versiones que se implantó elimina las deficiencias que presentaba el software anterior.
- Se facilitó el trabajo de los desarrolladores permitiendo el flujo de información entre los mismos.
- Se eliminó la dependencia de la red, haciendo que cada copia sea un repositorio completamente funcional.

4.4 Elaboración de la encuesta

Una vez terminada la implantación del controlador de versiones Bazaar en el proyecto BioSyS del Polo de Bioinformática, se elaboró un cuestionario, para conocer el grado de aceptación que tuvo dicho controlador.

Para la presentación del cuestionario se tuvieron en cuenta preguntas de conocimientos que se centran principalmente en los objetivos que debe cumplir la propuesta presentada, además de permitir algunas respuestas abiertas con posibilidad de emitir criterios personales.

La realización del temario fue preparado con la intención de facilitarles la solución de las preguntas presentas a los desarrolladores. La encuesta se puede encontrar en los anexos de este documento. (Anexo7).

4.5 Resultado de la encuesta

Con el fin de validar la propuesta del nuevo controlador de versiones se aplicó una encuesta a 20 desarrolladores del proyecto BioSyS y a una serie de estudiantes del proyecto FreeViews de la facultad 10 que utilizan el mismo, en total 40 personas. A continuación se muestra la relación de las repuestas de cada pregunta.

El procesamiento y análisis del cuestionario permitió valorar el nivel de conocimiento de los encuestados en las preguntas 1, 3 y 5 en cuanto a los controladores de versiones que dominan. Obteniendo como resultado que la mayoría tiene conocimiento de más de un software de este tipo, también se demuestra una vez más las insuficiencias que presenta Subversion, donde el cien por ciento de los desarrolladores las marca. En el punto cinco se puede apreciar que la propuesta realizada constituye una mejora para el control de versiones en el proyecto y goza de la aceptación de los desarrolladores. Ver tabla 1, 2, 3.

1-Marque con una X los controladores de versiones que conoce		
Controlador	**Cantidad**	**%**
CVS	12	30
Plastic SCM	2	5
Subversion	40	100
SourceSafe	5	12.5
Darcs	7	17.5
Git	25	62.5
Mercurial	17	42.5
Bazaar	40	100

Tabla #1.

3-Marque con una X las deficiencias que encuentra al uso de Subversion como controlador de versiones.		
Deficiencias	**Cantidad**	**%**
Dependencia de la red	40	100
Repositorio centralizado	40	100
Imposibilita el intercambio de información entre los desarrolladores	40	100

Tabla #2.

5-Marque con una X las potencialidades que le brinda el Controlador de Versiones Bazaar.		
Potencialidades	**Cantidad**	**%**
Elimina la dependencia de trabajar con conexión permanente a un servidor central.	40	100
Fácil de usar.	11	27.5
Se adapta a las necesidades de los desarrolladores personalizando sus flujos de trabajo.	20	50
Ejecuta los comandos con rapidez.	40	100
Presenta amplia documentación y manuales de usuario.	22	55

Tabla #3.

En los puntos 2, 4, 6 y 7 se les pide a los desarrolladores que valoren mediante preguntas cerradas la importancia que representa llevar el control de versiones en el desarrollo de software, pudiéndose apreciar en las repuestas que el cien por ciento considera de gran interés el uso de estas herramientas. También se comprobó que la nueva propuesta tuvo gran aceptación, ya que brinda facilidad de intercambio de información y elimina las deficiencias de Subversion. El punto seis refleja la gran acogida que tuvo la interfaz gráfica implementada dentro del proyecto.

¿Considera importante la realización del control de versiones de un proyecto de software?		
Si	**No**	**%**
40	0	100

Tabla #4.

¿Considera usted que el Controlador de Versiones Bazaar facilita el intercambio de información entre los desarrolladores del proyecto?		
Si	**No**	**%**
40	0	100

Tabla #5.

¿El prototipo funcional de aplicación desarrollado muestra una interfaz amigable?		
Si	**No**	**%**
31	9	77.5

Tabla #6.

¿Considera usted que con la implantación del Controlador de Versiones Bazaar en el proyecto se solucionan las deficiencias que presentaba Subversion?		
Si	**No**	**%**
40	0	100

Tabla #7.

En el punto 8, cuando se les preguntó a los desarrolladores su valoración sobre el control de versiones dentro del proyecto y qué cambios querían sugerir, la mayoría declaró que la herramienta implantada proporcionaba nuevos mecanismo para mejorar el trabajo en equipo. Además cuenta con un alto grado de adaptabilidad de sus flujos de trabajo al proyecto, ya que, aunque se trabaje de forma individual, en parejas o en equipo, estaban identificados con ellos. También se recogió la sugerencia de mostrar el modo de funcionamiento de otros controladores de versiones distribuidos para ampliar sus conocimientos al respecto.

Conclusiones

- A partir de un estudio abarcador de los controladores de versiones distribuidos más usados en la actualidad, por las potencialidades presentadas, se determinó la utilización de Bazaar en el Polo de Bioinformática.
- En aras de brindar una mayor comprensión y facilidad de uso, se realizó un Manual de Usuario del controlador de versiones Bazaar, el cual contiene instrucciones de instalación y comandos básicos.
- En la propuesta se implementó un prototipo funcional de aplicación proporcionando una interfaz gráfica al controlador de versiones, mejorando de esta forma, la apariencia y permitiendo una visibilidad clara y entendible.
- El software fue implantado en el Polo de Bioinformática, donde se verificó su correcto funcionamiento en el proyecto BioSyS.
- Para concluir la investigación se realizó un total de 40 encuestas a los integrantes de los proyecto BioSyS y FreeViews, para verificar el nivel de aceptación del nuevo controlador de versiones Bazaar.

Recomendaciones

Después de haber logrado los objetivos que se trazaron al inicio de este trabajo los autores recomiendan:

- Extender el uso de Bazaar a otros proyectos del Polo de Bioinformática.
- Desarrollar la interfaz desktop de la aplicación.
- Desarrollar una interfaz web.
- Continuar con la investigación para garantizar nuevas versiones y mejoras de la aplicación.

Bibliografía

1. Funcionamiento general de las herramientas colaborativas. [En línea] 28 de Septiembre de 2006. [Citado el: 20 de enero de 2009.] http://patob2000.wordpress.com.

2. Glosario sobre Linux. [En línea] [Citado el: 2 de enero de 2009.] http://www.escomposlinux.org/glosario.

3. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico 5ta edición.* 2005.

4. CVS. [En línea] [Citado el: 10 de enero de 2009.] http://www.nongnu.org.

5. **Wheeler, David A.** Free Software (OSS/FS). Software Configuration Management (SCM) Systems. [En línea] 10 de abril de 2004. http://www.dwheeler.com.

6. **Méndez, Salva G.** Control de Versiones. [En línea] [Citado el: 25 de febrero de 2009.] http://www.sgmendez.com.

7. Sistemas de Control de Versiones: Centralizados o Distribuidos. [En línea] [Citado el: 12 de Enero de 2009.] http://www.bosqueviejo.net.

8. Introduccion a los sistemas de control de versiones. [En línea] [Citado el: 15 de Febrero de 2009.] http://www.lug.fi.uba.ar.

9. **Lucarella, Leandro y Bertogli, Alberto.** Mitos y Verdades. [En línea] 22 de septiembre de 2006. http://www.lug.fi.uba.ar.

10. **Palomar, David.** Sistema de Control de Versiones Distribuido. [En línea] 17 de mayo de 2008. http://adamaconsulting.blogspot.com/2008/05/sistemas-de-control-de-versiones.html.

11. **García, Luis.** Sistema de control de versiones: Subversion. [En línea] [Citado el: 15 de febrero de 2009.] http://observatorio.cnice.mec.es.

12. **Sussman, Ben Collins, Fitzpatrick, B.W. y Pilato, C.M.** Control de versiones con Subversion. [En línea] 2004. [Citado el: 20 de febrero de 2009.] http://svnbook.red-bean.com. ISBN 0-596-00448-6.

13. **O'Sullivan, Bryan.** Control Distribuido de Revisiones con Mercurial. [En línea] [Citado el: 25 de febrero de 2009.] http://devnull.li/libromercurial/onepage/hgbook.html.

14. **Hamano Junio C, y un grupo de desarrolladores.** Git the fast version control system. [En línea] [Citado el: 3 de marzo de 2009.] http://git-scm.com .

15. Bazaar. [En línea] [Citado el: 5 de marzo de 2009.] http://bzr.cvn.org.

16. Glosario Salesiano. [En línea] http://www.edusal.cl/moodle/mod/glossary.

17. ¿Qué es OpenUp? [En línea] 2 de octubre de 2007. http://www.cendesi.com/.

18. Sitio de Metodología de Desarrollo. Ayuda extendida de OpenUp. [En línea] http://10.34.20.5:5800/OpenUp.

19. **Molpereces, Alberto.** Procesos de desarrollo: RUP, XP. [En línea]

20. **Kuchling, Andrew.** Python. [En línea] Septiembre de 2006. http://svn.python.org.

21. Python. [En línea] http://python.org .

22. NetBeans. [En línea] http://www.netbean.org.

23. Eclipse. [En línea] [Citado el: 15 de marzo de 2009.] http://www.eclipse.org.

24. Visual Paradigm. [En línea] http://www.visual-paradigm.com.

25. Sitio de la asignatura Ingenieria de software 1. *Conferencia 3 FT Requerimientos.* [En línea] [Citado el: 15 de Mayo de 2009.] http://teleformacion.uci.cu.

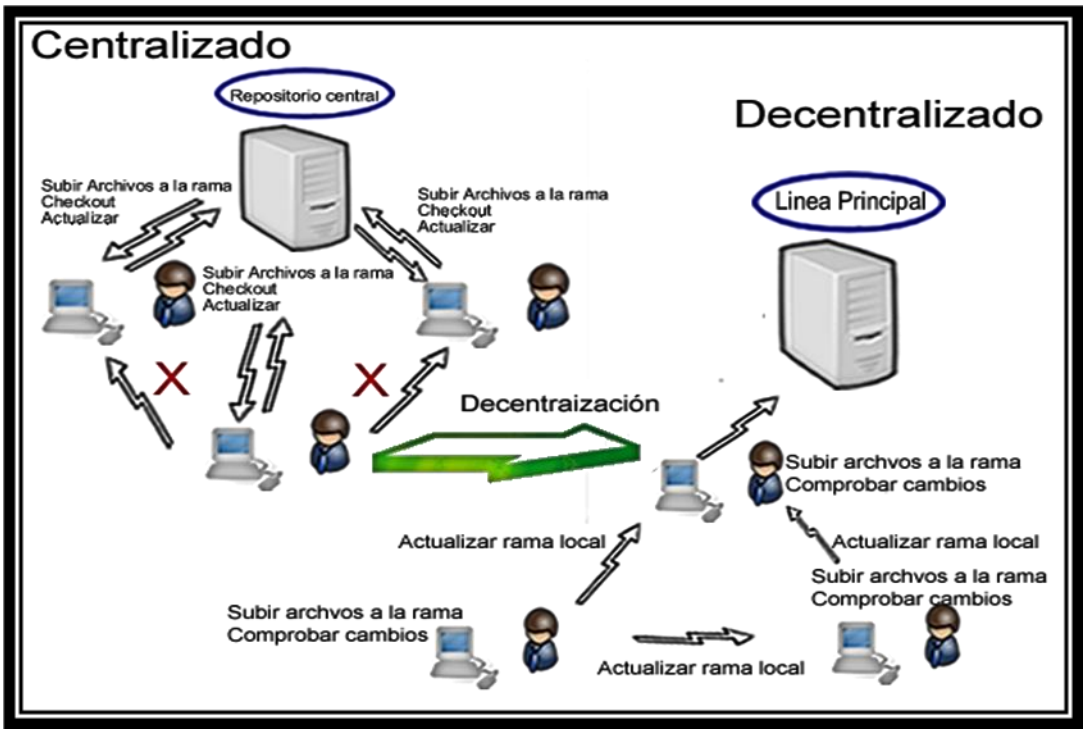
26. **Berrueta Muñoz, Diego.** Herramientas Colaborativas. [En línea] http://www.asturlinux.org..

27. **Labra Gayo, José Emilio, Fernández Lanvin, Daniel y Cernuda del Río, Agustín.** Una Experiencia de aprendizaje basado en proyectos utilizando herramientas colaborativas de desarrollo de software libre. [En línea] http://www.di.uniovi.es..

28. Perforce Software. [En línea] http://www.perforce.com/.

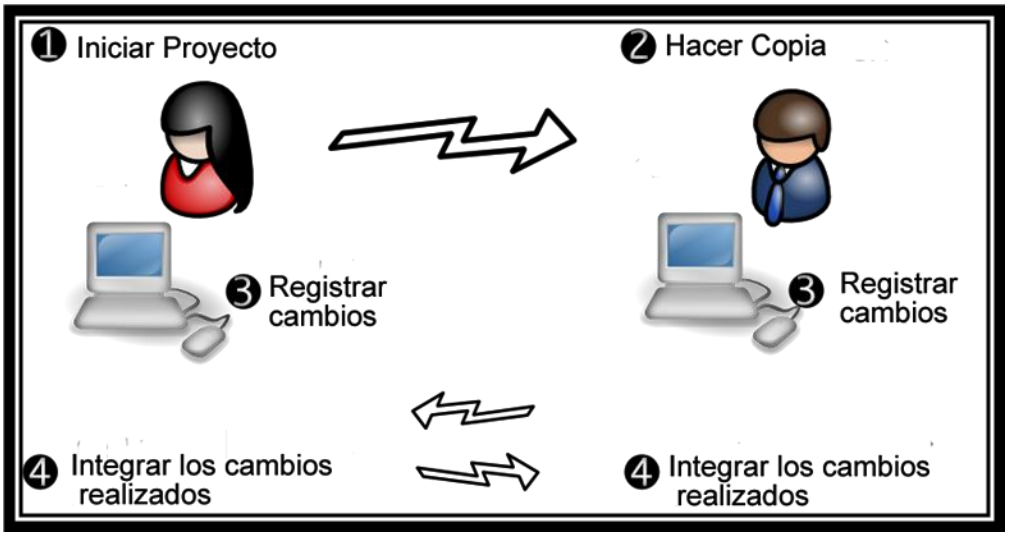
29. **Vidal Ledo, María y Concepción Báez, Carlos.** Herramientas para el trabajo colaborativo o sistema de gestión de contenidos. [En línea] http://bvs.sld.cu

Anexos

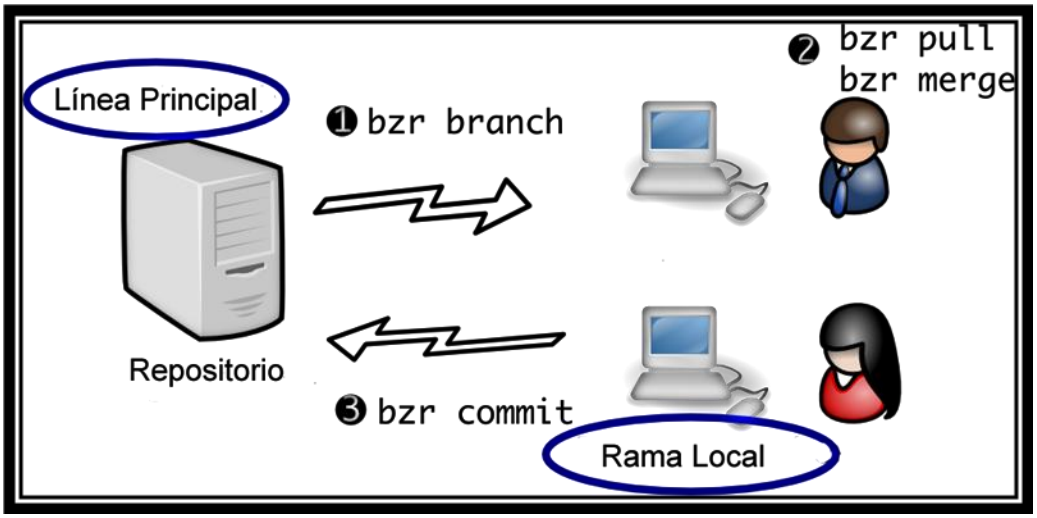


Anexo #1: Flujo de Centralizado contra Distribuido.

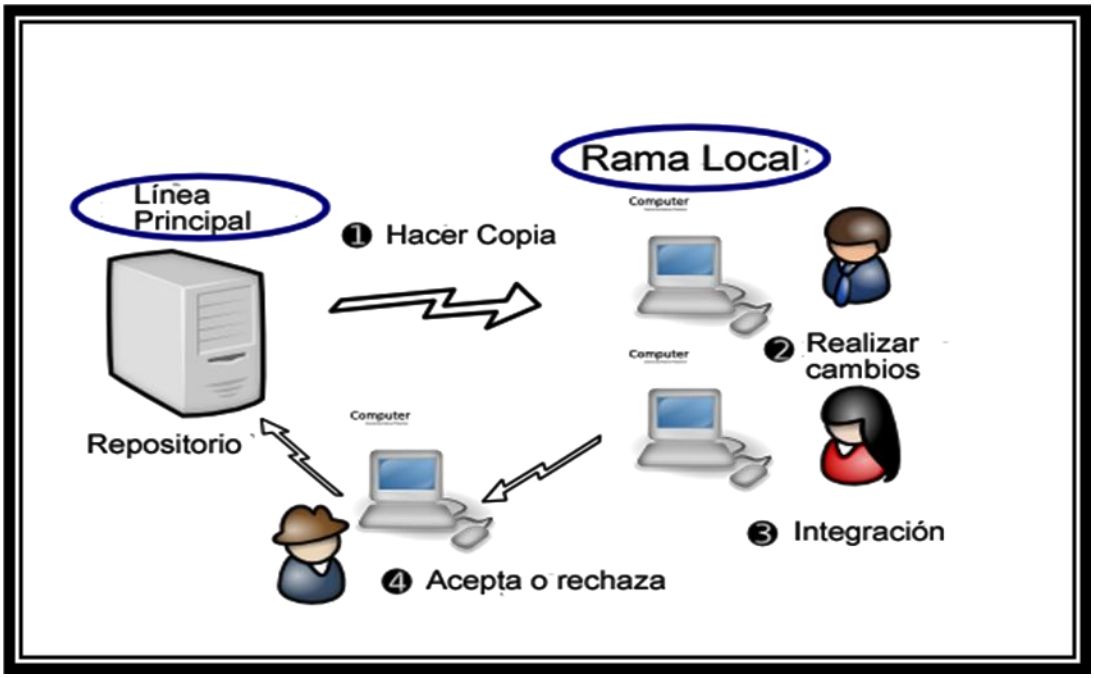
Anexo #2: Estrategia Solo.



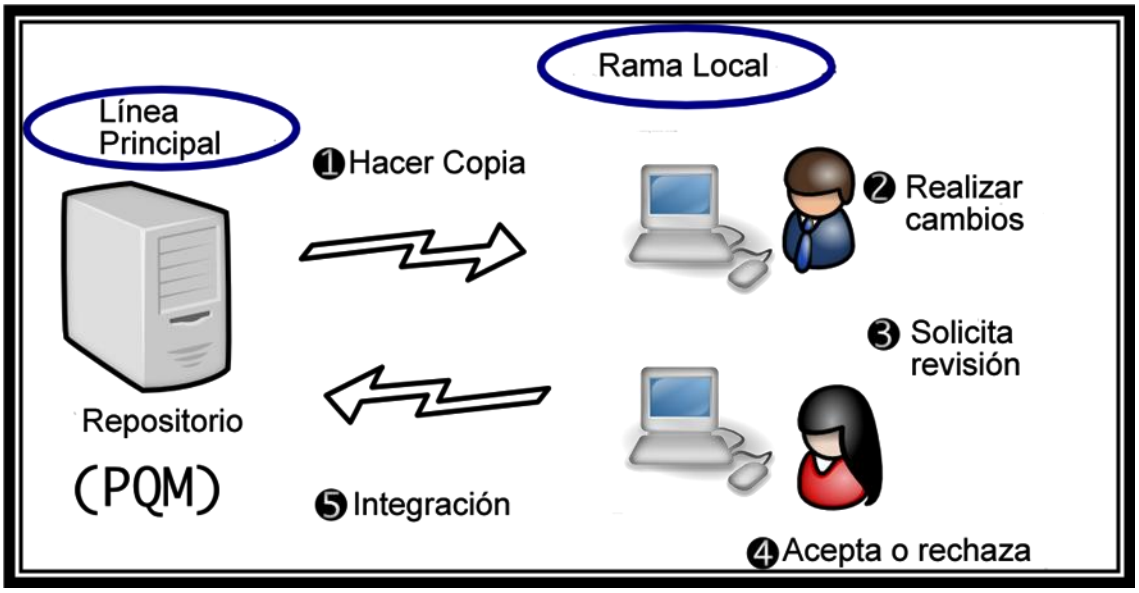
Anexo #3: Estrategia en Pareja.



Anexo #4: Estrategia descentralizada con línea principal compartida.



Anexo #5: Estrategia descentralizada con supervisor humano.



Anexo #6: Estrategia descentralizada con supervisor automático.

Anexo # 7 Encuesta aplicada a los desarrolladores

1. Marque con una X los controladores de versiones que conoce

   __SVC __Plastic SCM

   __Subversion __Git

   __ SourceSafe __Mercurial

   __Darcs __Bazaar

2. ¿Considera importante la realización del control de versiones de un proyecto de software?

   Si__ No___

3. Marque con una X las deficiencias que encuentra al uso de Subversion como controlador de versiones

   __Dependencia de la red

   __Repositorio centralizado

   __Imposibilita el intercambio de información entre los desarrolladores

4. ¿Considera usted que el Controlador de Versiones Bazaar facilita el intercambio de información entre los desarrolladores del proyecto?

   Si___ No___

5. Marque con una X las potencialidades que le brinda el Controlador de Versiones Bazaar.

   ___Elimina la dependencia de trabajar con conexión permanente a un servidor central.

   ___Fácil de usar.

   ___Se adapta a las necesidades de los desarrolladores personalizando sus flujos de trabajo.

   ___Ejecuta los comandos con rapidez.

   ___Presenta amplia documentación y manuales de usuario.

6. ¿El prototipo funcional de aplicación desarrollado muestra una interfaz amigable?

   Si___ No___

7. ¿Considera usted que con la implantación del Controlador de Versiones Bazaar en el proyecto se solucionan las deficiencias que presentaba Subversion?

   Si___ No___

8. Escriba al menos 3 sugerencias de cómo le gustaría que se realizara el control de versiones dentro del proyecto (puede ser tanto el uso de determinado software como sugerir diferentes flujos de trabajo)

Anexo # 8 Entrevista realizada a integrantes del proyecto BioSyS del Polo de Bioinformática

Objetivo: Constatar el conocimiento que poseen los integrantes del proyecto BioSyS sobre las limitantes que presenta Subversion.

Pregunta:

¿Qué deficiencias considera usted que posee el controlador de versiones Subversion dentro del proyecto? Argumente.

Glosario de Términos

Código Fuente: conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar un programa informático.

Commit: escribe o integra los cambios hechos a la copia local sobre el repositorio.

Directorio: almacena información acerca de los archivos que contiene.

Fichero: conjunto de información que se almacena para consultarse o utilizarse posteriormente.

Flujo de trabajo: estudio de los aspectos operacionales de una actividad de trabajo.

Hardware: partes físicas y tangibles de una computadora.

Línea Principal: lugar donde se almacena la información actualizada e integrada del proyecto.

Repositorio: lugar en el que se almacenan los datos actualizados e históricos, a menudo en un servidor. A veces se le denomina depósito. Puede ser un sistema de archivos en un disco duro, un banco de datos.

Rama: Línea de desarrollo. Lugar donde se almacenan un conjunto de revisiones del proyecto.

Software: conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Servidor: Una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.