REVISIÓN DE LOS SISTEMAS DE CONTROL DE VERSIONES UTILIZADOS EN EL DESARROLLO DE SOFTWARE

Edgar Tello-Leal **Claudia M. Sosa R.** **Diego A. Tello-Leal**
Universidad Autónoma de Tamaulipas, México *University of Southampton, UK*
etello@uat.edu.mx *clauqueen1@gmail.com* *datl1g11@soton.ac.uk*

(Tipo de Artículo: **Investigación**. Recibido el 14/05/2012. Aprobado el 26/06/2012)

RESUMEN
Uno de los retos a los que se enfrentan los desarrolladores de software es generar productos eficientes y de calidad sin sacrificar tiempo o costos. Este objetivo sólo se alcanza si los actores involucrados en tal proceso pueden disponer de toda la información relacionada con el proyecto. Los sistemas de control de versiones son aplicaciones que ayudan al proceso de desarrollo de software, facilitando la gestión del control de versiones de los archivos de código fuente generados por los desarrolladores, proporcionando herramientas para la fusión y generación de una nueva versión de un proyecto, permitiendo que múltiples desarrolladores trabajen en el mismo proyecto sin ocasionar pérdida de datos o bloqueos de archivos. Además, permiten recuperar archivos generados previamente, los cuales pueden ser utilizados para solucionar errores del sistema. En el presente trabajo de investigación se presenta una revisión de las principales aplicaciones de software disponibles para la gestión del control de versiones con un enfoque hacia su utilización en el desarrollo de software. Adicionalmente, se analiza su funcionamiento de acuerdo al método de administración de la información contenida en los repositorios, describiendo el proceso de creación, actualización y generación de versiones de archivos de código almacenados en los repositorios.

Palabras clave
Desarrollo de software, desarrolladores, sistemas de control de versiones, SCV centralizados, SCV distribuidos.

A REVIEW OF VERSION CONTROL SYSTEMS USED IN SOFTWARE DEVELOPMENT

ABSTRACT
Nowadays, one of the main challenges faced by software developers is creating efficient and quality products without impairing time or costs. This goal only can be achieved if people involved in the process can have all the information related to the project. Version control systems are applications that contribute in the software development process; they make easier the management of version control of source code files produced by developers, they also provide tools for the fusion and generation of a new project version thus allowing that different developers could work on the same project without causing data loss or blocking the access to files. Additionally, version control systems allow retrieving previously generated files, which can be used to fix system errors. In this work we present a review of the main software applications available for version control management from the point of view of their use in software development. Control version systems are also analyzed according to the management method of the information stored in repositories, by describing the process of creating, updating, and generating a new version of source code files stored in there.

Keywords
Centralized VCS, developer, distributed VCS, software development, version control systems.

UNE RÉVISION DES SYSTÈMES DE GESTION DE VERSION UTILISÉS EN DÉVELOPPEMENT LOGICIEL

RÉSUMÉ
Une des challenges qu'affrontent les développeurs logiciels est de créer des produits efficaces et de qualité sans affecter temps ou coûts. Cet objectif est obtenu si les personnes impliquées peuvent disposer de toute l'information liée au projet. Les systèmes de gestion de version sont des applications qu'aident au processus de développement logiciel, en facilitant la gestion du control des versions des fichiers de code source produit par les développeurs, en fournissant des outils pour la fusion et création d'une nouvelle version d'un projet, en permettant à plusieurs développeurs de travailler sur le même projet sans perte des données ou problèmes d'accès aux fichiers. En outre, les systèmes de gestion de version permettent de récupérer des fichiers déjà produits, lesquels peuvent être utilisés pour résoudre des erreurs systèmes. Dans cet article on présente une révision des principales applications logicielles disponibles pour la gestion du contrôle de version d'après son utilisation en développement logiciel. Les systèmes de gestion de version sont analysés selon la méthode de gestion de l'information qui est contenue dans les entrepôts, en décrivant le processus de création, actualisation et génération de versions de fichiers de code conservés.

Mots-clés
Développement logiciel, développeurs, systèmes de gestion de version, SGV centralisés, SGV distribués.

1. INTRODUCCIÓN

En la actualidad, la industria del software juega un papel cada vez más importante para la economía global. El software ha transformado los procesos de control de la mayoría de los servicios de los cuales dependemos. Cada día surgen más y mejores tecnologías y con ellas novedosas aplicaciones, generando nuevos retos para los implicados en los procesos de software.

Un sistema de software se describe como diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios web que permitan a los usuarios descargar información de productos recientes [1]. Por otro lado, en [2] se define software como un conjunto de actividades, métodos, prácticas y transformaciones que las personas utilizan para desarrollar y mantener el software y los productos asociados —proyectos, documentación de diseño, código, casos de prueba y manuales de usuario—.

En el área de desarrollo de software el disponer de una herramienta de control de versiones que permita el registro de las modificaciones que se hayan realizado a los programas o documentos, y que proporcione las versiones previas del software, puede evitar conflictos al momento de generar un producto final.

El control de versiones es el proceso de almacenar y recuperar cambios de un proyecto de desarrollo [3]. Los sistemas de control de versiones SCV, permiten disponer de una versión anterior para corregir errores o actualizar funciones. Dentro de sus funcionalidades está el conservar las versiones que se hayan generado a través del tiempo, así como los diferentes archivos que integran el proyecto en cuestión, uniendo en forma automática las aportaciones de los integrantes de un equipo de trabajo [4]. Los SCV pueden ser utilizados en muchos entornos, tales como el gestionar las versiones de documentos generados por procesadores de texto, presentaciones multimedia, archivos del sistema, correo electrónico y código fuente, por mencionar algunos [3].

En el entorno de desarrollo de software es recomendable contar con un mecanismo que permita coordinar las actividades y resultados de todos los desarrolladores involucrados en tal proceso. Los SCV permiten que un proyecto pueda avanzar con varias versiones al mismo tiempo y generar informes que muestren los cambios entre las etapas del proyecto [3].

Los SCV también son identificados como sistemas de control de código fuente (*Source Control Code Systems*, SCCS) [5], sistemas de gestión de control de código (*Source Control Management Systems*, SCMS) [6], sistemas de control de revisiones (*Revision Control Systems*, RCS) [6] o gestión de configuración de software (*Software Configuration Management* SCM) [7]. En el presente trabajo de investigación se hace referencia a los SCV enfocándolos exclusivamente a los sistemas, herramientas o aplicaciones desarrolladas para el control de versiones en el ambiente de desarrollo de software.

Los principales beneficios de utilizar una herramienta SCV son [6]: (1) cualquier revisión almacenada de un archivo puede ser recuperada para visualizarse o modificarse, (2) pueden desplegarse las diferencias entre distintas versiones, (3) las correcciones pueden ser creadas automáticamente, (4) múltiples desarrolladores pueden trabajar simultáneamente en el mismo proyecto o archivo sin pérdida de datos, (5) los proyectos pueden ser divididos para permitir el desarrollo simultáneo en varias versiones, estas divisiones pueden ser fusionadas para alcanzar el objetivo principal del desarrollo y 6) el desarrollo distribuido, es soportado a través de las redes de datos con diferentes mecanismos de autentificación. Estos beneficios favorecen la participación simultánea de los desarrolladores enriqueciendo el proceso de desarrollo de software, tal como se afirma en [8]: el control de versiones es una herramienta para manejar información que pueda estar disponible de manera colectiva y que se modifique constantemente.

Por lo expuesto anteriormente, en el presente trabajo de investigación se revisan las principales características de los SCV, analizando su clasificación en base a su entorno con enfoques centralizados y distribuidos. Además se detalla el funcionamiento y operación de los sistemas de control de versiones, describiendo los principales desarrollos de SCV de tipo abierto (*Open-Source*) y propietario, manteniendo una orientación hacia la aplicación del control de versiones en el desarrollo de software.

2. FUNCIONAMIENTO DE LOS SISTEMAS DE CONTROL DE VERSIONES

Los SCV son aplicaciones que guardan en repositorios las versiones de un software generadas en el transcurso de su desarrollo y evolución. Otero [9] describe a estos sistemas como los que se encargan de gestionar los diferentes estados por los que pasa una aplicación durante todo el periodo de su desarrollo, guardando un historial con todos los cambios realizados entre las versiones. Esto representa una importante ventaja para los desarrolladores, ya que estas versiones pueden estar disponibles para más participantes distribuidos geográficamente, propiciando que los actores (*stakeholders*) puedan contribuir en forma organizada, enriqueciendo y agilizando los procesos de desarrollo, sin sacrificar la calidad o elevar los costos [10].

Adicionalmente, el SCV almacena en un repositorio la fecha y hora de los cambios, así como el nombre del desarrollador que realizó las modificaciones, destacando las diferencias entre los archivos antes y después de los cambios. El contar con esta información facilita el reconstruir archivos previos cuando sea necesario, así como disponer en cualquier momento de las diferentes versiones del código fuente

[11]. En la Figura 1 se muestran las principales funciones de un SCV, destacando la gestión de versiones, la creación de división de trabajo y el almacenamiento del historial de cambios.

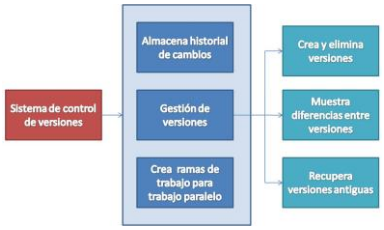

Fig. 1: Funcionamiento de un sistema de control de versiones

3. CLASIFICACION DE LOS SCV

Por la forma en que la información contenida en los proyectos es compartida y manipulada, los SCV se clasifican en centralizados y distribuidos. Los sistemas centralizados se caracterizan por contar con un servidor central de donde los desarrolladores toman información de alguna versión del proyecto, la manipulan y al finalizar el proceso de desarrollo, la actualizan en el servidor central. Los sistemas distribuidos no necesitan un servidor central para almacenar la información, sino que pueden disponer de alguna versión y trabajar localmente con la información, generando nuevas versiones, sin necesidad de almacenar la versión resultante en un servidor central. A continuación se analizan cada una de esas clasificaciones.

3.1 Sistemas de control de versiones centralizados

En un SCV centralizado todas las diferentes versiones de un proyecto están almacenadas en un único repositorio de un servidor central. Para que los desarrolladores puedan acceder a esas versiones o códigos fuente deben solicitar al SCV una copia local, en la cual realizan todos los cambios necesarios, y al finalizar recurren nuevamente al SCV para que almacene las modificaciones realizadas como una nueva versión. Es entonces cuando esa versión generada estará a disposición de los demás desarrolladores, al igual que las versiones anteriores.

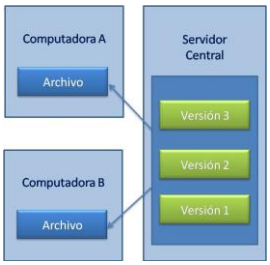

Fig. 2: Funcionamiento de un SCV centralizado

En la Figura 2 se muestra el funcionamiento de un SCV centralizado. Todos los desarrolladores trabajan sobre una rama o ramas (*branches*) en el mismo servidor central que aloja la última versión del proyecto. Este tipo de SCV requiere tener conexión de red para realizar las actualizaciones del servidor central [9]. Las principales aplicaciones de tipo SCV centralizado son el CVS [12] y Subversion [13].

- **Concurrent Version System (CVS).** El CVS [12] fue por mucho tiempo la principal herramienta para el control de versiones en ambientes Open-Source. Mediante su operación en red ha soportado que múltiples desarrolladores, dispersos geográficamente, puedan compartir sus aportaciones favoreciendo el trabajo colaborativo a través de una arquitectura cliente-servidor. De acuerdo con Lasa [4] los CVS se utilizan para gestionar los cambios al código fuente del proyecto, de modo que varios desarrolladores puedan trabajar de manera coordinada sobre el mismo código fuente. Pero estos sistemas no se limitan al código fuente, también se pueden usar para todo tipo de documentos o archivos que estén expuestos a sufrir cambios y se requiere conservar sus diferentes versiones, o en situaciones donde varias personas trabajen en un mismo proyecto [3].

  En la Figura 3 se muestra el proceso de actualización de un repositorio utilizando la aplicación CVS. Del repositorio base, el programador A y el programador B obtienen cada uno una copia de trabajo del mismo proyecto (*checking out*). Al disponer cada desarrollador de una copia independiente de los archivos requeridos, puede modificarlos libremente, sin existir ningún tipo de restricción. Al finalizar el proceso de cambios, los desarrolladores llevarán a cabo el registro de los cambios (*commit*); de existir una situación de conflicto (*conflict*) generada por intentar modificar la misma línea o área de texto por ambos desarrolladores, CVS emitirá una notificación de error, colocando un "marcador de conflicto" en la ubicación que está provocando dicha situación. En este punto, los desarrolladores involucrados buscarán una solución a este escenario. Por el contrario, si los cambios realizados por ambos desarrolladores no generan conflictos, se llevará a cabo la actualización en el repositorio, generándose automáticamente un número de versión y almacenando información adicional (*log-message*), que incluye datos como la fecha y el nombre de los autores de los cambios [14].

Fig. 3: Proceso de actualización de un repositorio utilizando CVS

- **Subversion.** El sistema Subversion [13] es un SCV de código abierto (open-source), que maneja los cambios realizados tanto en archivos como en directorios; esto permite recuperar versiones anteriores de sus datos o examinar la historia de cómo cambiaron sus datos. Por otro lado, ofrece una estructura de árbol de directorios en un

repositorio central; el repositorio es como un servidor de archivos excepto que recuerda todos los cambios realizados a sus archivos y directorios. Subversion permite conservar distintas versiones de los directorios, característica que no soporta su antecesor CVS [4]. Además Subversion soluciona la mayoría de las deficiencias presentadas por CVS.

Subversion permite al usuario obtener del repositorio una copia del proyecto, con la cual los desarrolladores trabajan en paralelo realizado sus cambios, para finalmente integrar sus copias en una versión final [8]. Subversion utiliza un modelo copiar-modificar-fusionar (*copy-modify-merge*) como alternativa para evitar el bloqueo de archivos en el proceso de actualización. En este modelo cada usuario se conecta al repositorio del proyecto y crea una copia personal (copia espejo de los archivos y directorios contenidos en el repositorio) en la cual trabajará. Después los usuarios pueden trabajar simultáneamente modificando sus copias privadas y personales. Finalmente las copias privadas son fusionadas en una nueva versión. Subversion facilita el proceso de fusión de las diferentes copias pero el desarrollador es el responsable de que se realice correctamente esta parte del proceso [15]. La forma de operar de Subversion mejora la productividad debido a que los archivos están siempre disponibles, al no bloquearlos cuando se pretendan actualizar por varios desarrolladores al mismo tiempo. La Figura 4 muestra un ejemplo del proceso de actualización en un repositorio por parte de dos desarrolladores, basado en la propuesta descrita en [15].

Fig. 4: Proceso de actualización de los archivos en un repositorio utilizando Subversion

Subversion ofrece importantes ventajas con respecto a su antecesor CVS [15]:

- **Versionado de directorios**. CVS sólo controla la historia de los archivos en forma individual. En cambio, Subversion implementa un sistema de archivos "virtual" de versiones que sigue los cambios de la estructura de directorios. Tanto a los directorios como a los archivos se les asigna una versión.

- **Control de historial**. Dado que CVS está limitado al control de versiones de archivos, no permite copiarlos y/o renombrarlos. Además, CVS no puede reemplazar un archivo versionado con otro que lleve el mismo nombre, sin que el nuevo archivo herede el historial del anterior, que tal vez sea completamente distinto al nuevo archivo. En cambio, con Subversion, se puede agregar, eliminar, copiar y renombrar archivos y directorios. Adicionalmente, cada archivo nuevo que es añadido inicia con un historial limpio o vacío.

- **Atomicidad**. Un conjunto de modificaciones sólo puede integrarse al repositorio si está totalmente completo. Esto permite a los programadores realizar cambios como unidades lógicas, lo cual evita los problemas que pueden ocurrir cuando sólo una parte de las modificaciones son enviadas al repositorio. El término “atómico” implica la indivisibilidad e irreductibilidad del proceso, ya que éste debe realizarse en su totalidad o de lo contrario no realizarse.

- **Versión en los Metadatos**. Cada directorio y archivo tienen un conjunto de propiedades asociadas, las cuales se pueden crear y almacenar arbitrariamente.

- **Conexión en diferentes capas de red**. Los accesos al repositorio se pueden implementar mediante diferentes mecanismos de red. Subversion puede conectarse a un servidor Apache usando el protocolo HTTP. Esto dota a Subversion de una ventaja en estabilidad e interoperabilidad, además de las características que provee el servidor Apache, tales como autenticación, autorización y compresión de datos. También una versión ligera del servidor de Subversion está disponible, la cual utiliza un protocolo propio que puede encapsularse sobre el protocolo SSH.

- **Manejo de datos consistente**. Subversion identifica las diferencias de un archivo usando un algoritmo de diferenciación binario, que funciona exactamente igual en archivos de texto (legibles) y archivos binarios (ilegibles por los humanos). Ambos tipos de archivos se almacenan comprimidos en el repositorio y las diferencias se transmiten en ambas direcciones a través de la red.

- **Team Foundation Server.** Microsoft Visual Studio Team Foundation Server [16] es un software propietario cuya plataforma de colaboración se encuentra en el núcleo de la solución de Visual Studio para la administración del ciclo de vida de una aplicación. Team Foundation Server (TFS) proporciona servicios fundamentales como control de versiones, seguimiento de elementos de trabajo y errores, automatización de la compilación y almacén de datos. TFS es un sistema centralizado que aporta una serie de herramientas que permiten

colaborar y coordinar las tareas de un equipo de trabajo para llevar a cabo un proyecto, mejorando la comunicación entre los integrantes. Además, permite la creación de informes, implementación de metodologías de procesos, tendencia histórica y visibilidad del estado general del proyecto mediante un panel basado en etiquetas. Adicionalmente, se pueden visualizar métricas en tiempo real que alertan sobre potenciales problemas en etapas tempranas del desarrollo de software de modo que se puedan tomar las decisiones basadas en datos y aplicarse las correcciones adecuadas [16].

TFS ofrece interfaces para facilitar la administración y visualización de ramas y manejo de cambios. Su sistema de reportes está basado en Windows Workflow Foundation 4.0. Otra de las características importantes de TSF es el almacenamiento de los datos del control de versiones en un servidor de base de datos SQL.

3.2 Sistemas de control de versiones distribuidos

En el SCV distribuido cada desarrollador realiza una copia del repositorio de proyectos completo a su computadora, generándose un repositorio local del proyecto. Este repositorio local incluye todos los archivos del proyecto y el historial de cambios generado en previas actualizaciones. Cuando cada desarrollador realice modificaciones a los archivos de su repositorio local, el contenido de este repositorio se irá distanciando de los repositorios locales de otros desarrolladores, lo que provocará que se generen ramas de un mismo proyecto en los repositorios de cada desarrollador. De esta manera los desarrolladores pueden trabajar paralela e independientemente, guardando sus propias versiones en un repositorio local [9].

En la etapa del proceso de desarrollo del proyecto que requiera unir los repositorios locales de los desarrolladores, el SCV realizará una sincronización de copias, con lo cual se generará una nueva versión del proyecto conteniendo todos los cambios realizados por cada desarrollador. Este tipo de SCV mantiene un repositorio como referencia para realizar la sincronización de repositorios locales. En el SCV distribuido no es necesario mantener una conexión de red permanente a un servidor que contenga el repositorio de referencia, las conexiones de red por parte de los desarrolladores solo se requieren cuando se realiza el proceso de sincronización. Los principales SCV distribuidos de acuerdo a la clasificación presentada en [10] son GIT, Mercurial, BZR y BeetKeeper.

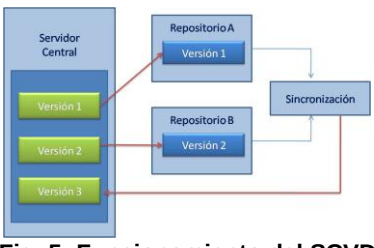

Fig. 5: Funcionamiento del SCVD

En la Figura 5 se muestra brevemente el funcionamiento de un sistema de control de versiones distribuido.

- **Git** [17]. Es uno de los SCV distribuidos más populares, inicialmente desarrollado para Linux. Git permite a varios programadores trabajar paralelamente con sus propias copias de trabajo obtenidas de un repositorio, como lo efectúan todos los SCV distribuidos. Git está compuesto de una estructura de tres secciones [9], como se describe en la Figura 6, las cuales son:

  - Directorio de Git. Es donde se guardan los objetos que mantienen el historial con los cambios que se han producido en el proyecto.

  - Directorio de trabajo. Contiene los archivos de la versión actual del proyecto sobre los que se realizan los cambios.

  - Área de preparación o índice. Es un archivo que incluye la información de los cambios que se van a enviar en la próxima confirmación.

Fig. 6. Estructura de trabajo de Git

Git utiliza ramas favoreciendo el trabajo paralelo sobre un mismo proyecto. En el momento de iniciar un repositorio se genera una rama maestra, de donde se extienden nuevas ramas que incluirán todo el historial del proyecto. Cuando se han realizado los cambios en una rama, permite que ésta se combine con otras ramas, uniéndose a la rama maestra (*merge*), integrando historiales y archivos de las ramas participantes; es entonces cuando se genera una nueva versión [9]. El trabajo en ramas de Git se ilustra en la Figura 7.

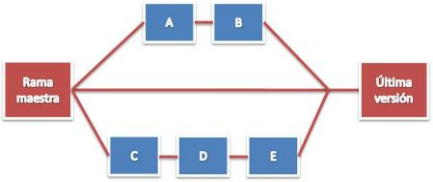

Fig. 7: Funcionamiento del trabajo en ramas de Git

El proceso de actualización de los archivos contenidos en un repositorio y la posterior generación de una nueva versión utilizando Git es el siguiente: cada desarrollador debe generar una copia del repositorio original en su computadora, creando un repositorio local. Este repositorio local contiene toda la información del historial de cambios y los archivos del directorio de trabajo. En esta etapa el desarrollador puede empezar a trabajar en el repositorio Git local, creando y modificando los archivos de acuerdo a sus requerimientos.

Para crear y almacenar una nueva versión el procedimiento es el siguiente: primeramente se debe consultar el estado del repositorio, con lo cual se obtiene un listado de los archivos que han sido modificados. Enseguida, se seleccionan los que se almacenarán en la nueva versión, esto es, los archivos que contienen cambios y se quieren incluir en la nueva versión. Posteriormente, se detallan los cambios que se han realizado en los archivos, esto servirá a los desarrolladores para identificar las versiones. Al finalizar estos pasos, Git almacenará en la sección "área de preparación" una nueva copia en el historial del proyecto con todos los cambios incluidos. Posteriormente, el desarrollador debe confirmar los cambios, con lo cual Git almacenará una nueva versión del proyecto en la sección "directorio de git" conteniendo los archivos que sufrieron cambios y, de los archivos que no fueron modificados solo guarda el enlace al archivo anterior, que ya se encontraba en el repositorio del proyecto.

En esta parte del proceso del funcionamiento de Git, los cambios y actualización de versión solo se han realizado en el repositorio local del desarrollador. Enseguida se debe realizar un proceso de fusión (*merge*), el cual consiste en combinar una o varias ramas de un proyecto en un repositorio local al repositorio origen o al repositorio de otros desarrolladores. La acción de *merge* realiza una combinación de los historiales y archivos de las dos ramas implicadas. Git detecta los cambios que existen en las dos ramas, combinándolas y generando una única versión con los cambios realizados en ambas ramas. Al efectuar el *merge* se cambian todos los archivos en la rama del proyecto destino, generándose una nueva versión.

- **Mercurial.** El SCV distribuido Mercurial [18] permite copiar un repositorio, conteniendo toda la historia del desarrollo del proyecto, esta copia del repositorio local funciona en forma independiente y autónoma sin requerir acceso a la red o a un servidor. La copia incluirá todos los archivos del proyecto y su historial. El repositorio local identifica la ubicación del repositorio origen, pero Mercurial no se comunicará con ese repositorio, a menos que el desarrollador lo realice. Mercurial tiene una interfaz de web de gran alcance que ofrece las siguientes funciones [6]: navegación sobre la estructura de repositorios, visualización del historial de cambios, despliegue del contenido de directorios y archivos, utilización de Atom y RSS para estar al tanto de los cambios en el repositorio, soporta usuarios remotos para copiar, modificar y actualizar repositorios.

  El sistema Mercurial permite, mediante una extensión *convert,* importar proyectos de aplicaciones como Subversion, CVS, Git, Darcs, convertiendo toda la historia del proyecto en uno nuevo en el repositorio Mercurial. Adicionalmente, la extensión *convert* permite exportar los cambios a Subversion, posibilitando el llevar a cabo en paralelo el desarrollo de un proyecto.

  Una de las características principales de Mercurial es su funcionamiento en línea de comandos. El proceso básico del funcionamiento de Mercurial es el siguiente: primero se debe de crear una copia local del repositorio (*hg clone*); el repositorio del proyecto local contendrá todos los archivos relacionados al proyecto con su registro histórico. En esta etapa el desarrollador puede iniciar los cambios o actualizaciones en los archivos del proyecto en su repositorio local, posteriormente el desarrollador puede generar una nueva revisión (*hg commit*) la cual incluye los archivos que han sido modificados. Enseguida el desarrollador debe autorizar los cambios, generándose una nueva versión en su repositorio local (*hg push*). Para integrar la nueva versión al directorio de trabajo y que pueda ser copiada por otros desarrolladores a sus repositorios locales se debe de realizar una actualización (*hg update*).

4. TRABAJOS RELACIONADOS

En los procesos de migración de plataforma de control de versiones, por ejemplo de CVS [12] a Subversion [13], normalmente los nuevos repositorios sólo contienen un resumen de la historia de los archivos de código fuente y del proyecto de desarrollo, lo cual ocasiona la pérdida de información histórica del desarrollo de software y en algunos casos puede limitar la re-utilización de código por falta de información de los cambios en los archivos. Para solucionar este problema y permitir proyectos de software más eficientes en [19] se proponen un enfoque para construir un sistema universal histórico de versiones, en el que múltiples sistemas de control de versiones se puedan conectar a los diferentes repositorios para recuperar información de los cambios en los archivos.

En [20] se presenta un enfoque que permite combinar datos de los sistemas de control de versiones con los sistemas de control de errores o fallas con el objetivo de proporcionar una visión completa sobre la evolución de un proyecto de software. Para lo cual propone utilizar una base de datos que contendrá información histórica de los dos sistemas y mediante un análisis encontrar los puntos de unión. Enseguida, pueden ser aplicadas consultas a los datos estructurados para obtener puntos de vista significativos que muestren la evolución de un proyecto de software. Estos puntos de vista permiten un razonamiento más preciso de los aspectos que han cambiado en el proyecto y se puedan anticipar los futuros cambios en el proyecto.

En [21] presentan un sistema que permite visualizar la evolución del desarrollo de software usando técnicas de grafos. El sistema propuesto extrae información de programas basados en Java almacenados en repositorios de CVS [12] y la muestra usando un visualizador basado en grafos. La información presentada permite a los programadores comprender la evolución de los programas, su estructura, los programadores que han participado en el desarrollo de ese programa, en qué parte del programa y durante

qué periodos, y las secciones del programa que se han comportado inestables y han requerido gran cantidad de cambios. Esta información proporcionada por el sistema es un complemento a la que generan las herramientas estadísticas como los buscadores de códigos y los analizadores estadísticos. En conjunto pueden ser de gran utilidad para los líderes de proyecto en ambientes de desarrollo de software.

En [22] se presenta un sistema que soporta la exploración visual de los repositorios de Subversion [13], así como su evolución con respecto al proceso de desarrollo y los cambios en el código fuente. Se hace uso de la estructura jerárquica de archivos para dar soporte a los usuarios en las tareas de navegación y orientación de manera que puedan obtener una visión de los datos en diferentes niveles de detalle (directorios, archivos y líneas de códigos). El enfoque utiliza diferentes técnicas de visualización lo que permite generar diversos tipos de gráficas, facilitando el análisis del progreso de los desarrollos de software.

En [23] se presenta una comparación entre Subversion [13] y Git [17], sistemas de control de versiones de tipo centralizado y distribuido respectivamente. Mediante el desarrollo e implementación de un escenario en paralelo, se detalla paso a paso la operación de los dos enfoques y se presenta un comparativo en cada una de las fases de la operación de los sistemas, describiendo sus funcionalidades y describiendo las ventajas que ofrece el sistema distribuido sobre el enfoque centralizado.

En [24] se presenta un escenario para implementar un sistema de control de versiones utilizando Mercurial, con el objetivo de incorporar la utilización de sistemas de control de versiones en los planes de estudio de ingeniería de sistemas, lo cual permitirá a los estudiantes trabajar en forma colaborativa con actividades y proyectos reales. Además, de conocer las expectativas prevalentes en las comunidades de desarrollo de software.

En [25] presentan un sistema de replicado de control de versiones basado en Git [17], el cual proporciona mecanismos de replicación de un repositorio global en combinación con repositorios descentralizados del lado del cliente. Este enfoque combina modelos de objetos fragmentados con anotaciones semánticas que permiten personalizar y optimizar los mecanismos de replicación. El sistema ofrece ventajas en la infraestructura de replicación, tales como la capacidad de proveer código de los clientes mediante un servicio remoto y el soporte para la optimización del protocolo que permite la interacción remota, lo que contribuye a generar un potente sistema de replicación de control de versiones con tolerancia a fallos.

En [26] se propone utilizar tecnologías de Web semántica para transformar los sistemas de control de versiones distribuidos en una Web semántica social. En la propuesta se hace uso de ontologías que permiten capturar y representar la historia de los cambios en los archivos y su posterior recuperación en escenarios de control de versiones basados en Git [17], Mercurial [18] y Bazaar [27]. Mediante este enfoque se permite a cada nodo de la red publicar grupos de datos semánticos. Enseguida, estos grupos de datos pueden ser consultados a través de enlaces transversales ejecutando "*queries*" basadas en cálculo de métricas y descubrimiento de nodos o clientes de la red semántica.

5. CONCLUSIONES

Los sistemas de control de versiones ofrecen un soporte muy importante en el proceso de desarrollo de software, coadyuvando en la gestión del control de versiones de los archivos de código fuente e impulsando el trabajo colaborativo al permitir que múltiples desarrolladores trabajen paralelamente en un mismo proyecto en forma autónoma e independiente.

Adicionalmente, los sistemas de control de versiones proporcionan los mecanismos para que cualquier versión de un proyecto pueda ser recuperada para visualizarse o modificarse, desplegando las diferencias entre las versiones existentes. Por otro lado, el desarrollo distribuido es soportado a través de redes de datos con diferentes mecanismos de autentificación.

En el presente trabajo de investigación se presentó un análisis de los sistemas de control de versiones, tanto "*open-source*" como propietarios, describiendo las principales funciones que ofrecen al momento de realizar una implementación y/o aplicación en el desarrollo de software.

Además se define el proceso de creación y actualización de repositorios, tanto en el enfoque centralizado como distribuido, detallándose el proceso de actualización de archivos de código fuente de acuerdo a cada una de las herramientas revisadas.

Cabe destacar que la presente investigación servirá como base teórica para el desarrollo de una herramienta para el control de versiones basado en servicios web que el grupo de investigación tiene planeado desarrollar en un futuro y cuyo fundamento se contempla dentro de las líneas de investigación y de los proyectos vigentes del cuerpo académico.

6. REFERENCIAS

[1] I. Sommerville. "Ingeniería del Software". Pearson Educación, 2005.

[2] F. Medina. "Marco Metodológico para la Mejora de la Eficiencia de Uso de los Procesos de Software". Tesis Doctoral, Universidad Carlos III de Madrid, España, 2010.

[3] J. Vesperman. "Essential CVS". O'Really Media Inc, 2007.

[4] M. Lasa. "Desarrollo de aplicaciones en entornos de software libre". Tesis de Maestría, Universitat Oberta de Catalunya, España, 2010.

[5] M. J. Rochkind. "The Source Code Control System", IEEE Transitions on Software Engineering, Vol. Se-1, No. 4, pp. 364-370, 1975.

[6] B. O'Sullivan. "Mercurial: The Definitive Guide". O'Really Media Inc, 2009.
[7] M. Koegel et al. "Comparing State- and Operation-based Change Tracking on Models". Proceedings 14th IEEE International Enterprise Distributed Object Computing, EDOC2010, pp. 163-172, 2010.
[8] F. Solsona & E. Viso. "Manual de supervivencia en Linux". Universidad Autónoma de México, Facultad de Ciencias, 2007.
[9] D. Otero. "Desarrollo de una aplicación Web para control de versiones de software". Tesis Doctoral, Universidad Carlos III de Madrid, España, 2011.
[10] B. Alwis & J. Sillito. "Why are Software Projects Moving from Centralized to Decentralized Version Control Systems?" Proceedings Workshop on Cooperative and Human Aspects on Software Engineering, ICSE2009. pp. 36-39, 2009.
[11] K. Hinsen; K. Läufer & G. K. Thiruvathukal. "Essential Tools: Version Control Systems". Journal of IEEE Computing in Science & Engineering, Vol. 11, No. 6, pp. 84-91, 2009.
[12] Concurrent Version System. Online [Feb. 2012].
[13] Apache Subversion. Online [Jan. 2012].
[14] M. Bar & K. Fogel. "Open Source Development with CVS". Paraglyph Press, 2003.
[15] B. Collins-Sussman; B. W. Fitzpatrick & C.M. Pilato. "Version Control with Subversion". O´Reilly Media, 2004.
[16] C. Menegay. "Using Source Code Control in Team Foundation". Microsoft Visual Studio 2005 Team System, 2005.
[17] Git. Online [Feb. 2012].
[18] Mercurial SCM. Online [Jan. 2012].
[19] A. Mockus. "Amassing and indexing a large sample of version control systems: towards the census of public source code history". Proceedings 6th IEEE International Working Conference on Mining Software Repositories, MRS '09", pp. 11-20, 2009.
[20] M. Fischer; M. Pinzger & H. Gall. "Populating a Release History Database from Version Control and Bug Tracking Systems". Proceedings International Conference on Software Maintenance, ICSM '03, pp. 23-32, 2003.
[21] C. Collberg et al. "A System for Graph-Based Visualization of the Evolution of Software". Proceedings 2003 ACM symposium on Software visualization, SoftVis '03, pp. 77-87, 2003.
[22] C. Müller et al. "Subversion Statistics Sifter". Lecture Notes in Computer Science Volume, Vol. 6455, pp. 447-457, 2010.
[23] C. Bird et al. "The Promises and Perils of Mining Git". Proceedings 6th IEEE International Working Conference on Mining Software Repositories, MRS '09", pp. 1-10, 2009.
[24] D. Rocco & W. Lloyd. "Distributed Version Control in the Classroom". Proceedings 42nd ACM technical symposium on Computer science education. SIGCSE'11, pp. 637-642, 2011.
[25] R. Kapitza; P. Baumann & H. P. Reiser. "Using Object Replication for Building a Dependable Version Control System". Distributed Applications and Interoperable Systems, Vol. 5053, pp.86-99, 2008.
[26] K. Aslan; H. Skaf-Molli & P. Molli. "Connecting Distributed Version Control Systems Communities to Linked Open Data". Proceedings the International Conference on Collaboration Technologies and Systems, CTS 2012, pp. 1-9, 2012.
[27] Bazaar. Online [Mar. 2012].