Los sistemas de control de versiones

The systems of control of versions

[*]**Luis Felipe Wanumen Silva**

Fecha de recepción: 7 de octubre de 2008
Fecha de aceptación: 26 de noviembre de 2008

Resumen

Se muestran los avances en cuanto a los sistemas de control de versiones que son día por día uno de los temas más apasionantes dentro del enorme campo de la ingeniería de software; también se pretende dar unas pautas sobre las características globales de los CVS, además de analizar con algo de detalle las perspectivas a futuro que se tienen con los CVS. Se trata al máximo de hacer algunas comparaciones entre los sistemas amparados bajo licencia GNU y algunos propietarios para mostrar varias de las características que ofrecen para implementar sistemas de control de versiones.

Palabras clave: ingeniería de software, control de versiones de software, sistemas de control de versiones, control de código fuente, control de versiones de documentación, gestión del cambio y configuraciones, cambios en el desarrollo de sistemas.

* Ingeniero de Sistemas, especialista en Ingeniería de Software de la Universidad Distrital Francisco José de Caldas. Docente de la Universidad Distrital Francisco José de Caldas, Facultad Tecnológica. Director del Grupo de Investigación Desarrollo de Herramientas para la Creación y Manipulación de Contenido XML. lwanumen@udistrital.edu.co

Abstract

The following work seeks to show the advances as for the systems of control of versions that are day per day one of the most exciting topics inside the enormous field of the Engineering of Software, it is also sought to give some rules on the global characteristics of the CVS, besides analyzing with something of detail the perspectives to future that you/they are had towards the CVS. It is important to notice that the present article tries in the maximum thing to make some comparisons among the systems aided under it licenses GNU and some proprietors to show some of the characteristics that they offer to implement systems of control of versions.

Key words: Engineering of software, control of versions of software, systems of control of versions, control of code source, control of versions of documentation, administration of the change and configurations, changes in the development of systems..

1. Marco teórico

1.1 Ciclos de desarrollo según el proceso unificado de desarrollo

El proceso unificado de desarrollo está compuesto por fases e iteraciones. Una fase es un intervalo de tiempo entres dos hitos importantes del proceso, durante la cual se cumple un conjunto bien definido de objetivos, se completan artefactos y se toman las decisiones sobre si pasar a la siguiente fase. Las cuatro fases del proceso unificado son:

- Iteración
- Elaboración
- Construcción
- Transición

Dentro de cada fase hay varias iteraciones que algunos denominan flujos de trabajo y que son:

- Modelo del negocio: describe la estructura y la dinámica de la organización.

- equisitos: describe el método basado en casos de uso para extraer los requisitos.

- Análisis y diseño: describe las diferentes vistas arquitectónicas.

- Pruebas: describe los casos de pruebas, los procedimientos y métricas para evaluación de defectos.

- Gestión de configuraciones: controla los cambios y mantiene la integridad de los artefactos de un proyecto.

- Gestión del proyecto: describe varias estrategias de trabajo en un proceso iterativo.

- Entorno: cubre la infraestructura necesaria para desarrollar un sistema.

1.2. Planteamiento del problema

Dentro del marco teórico plasmado anteriormente, vemos que la gestión de configuraciones cobra una especial importancia en el desarrollo de sistemas y obviamente, como casi todas las cosas, ha sido plasmada primero en forma teórica por expertos del área y luego algunas personas han tratado de dar soluciones reales en el área. Pues bien, la gestión de configuraciones es un área tratada durante años por la ingeniería de software, pero en el último tiempo se ha incrementado tanto su interés que surgieron teorías al respecto. La cuestión es que, como parte de la gestión de configuraciones, un punto bien importante a tratar es la gestión de versiones, cosa que hasta hace unos años se trataba solamente en forma teórica, pero afortunadamente en estos últimos cinco años se ha visto favorecida debido al surgimiento de sistemas que permiten a los desarrolladores de sistemas gestionar las versiones e incluso ir más allá y gestionar las subversiones. Obviamente los sistemas actuales que permiten la gestión de configuraciones son sistemas que tienen algunas limitaciones, pero que así mismo también brindan algunas facilidades, y este es precisamente el objetivo de este artículo: poder dar una idea mundial del estado en el que se encuentran dichos sistemas y poder hacer algunas predicciones de los posibles caminos que pueden tomar estos sistemas basándonos en las cuestiones que les hace falta mejorar a los sistemas actuales. También se pretende sacar algunas conclusiones sobre lo que son y lo que no son los sistemas de gestión de configuraciones.

1.3. Análisis crítico

Se han realizado algunos trabajos similares al actual, pero han sido desarrollados en su mayoría por personas que están inclinadas tecnológicamente a utilizar una u otra tecnología; por ejemplo Microsoft ha publicado algunos trabajos para hablar sobre la gestión de configuraciones y específicamente sobre la gestión del control de versiones en proyectos de desarrollo de sistemas, pero inmediatamente después habla sobre las herramientas que ofrece para apoyar dicha gestión. Esto obviamente se hace debido a que están inclinados por la utilización de dichas herramientas. En otros casos algunos fervientes amantes del software amparado por GNU muestran a los sistemas CVS de código abierto como los mejores, pero es bueno tener objetividad para hacer análisis de medios en los que no se subestime una u otra tecnología. En este sentido, el presente artículo espera ser de valiosa ayuda para quienes quieran comprender el estado actual de los sistemas de control de versiones.

1.4. Limitaciones del marco teórico

El tema de la gestión de configuraciones y específicamente la gestión de control de versiones tiene un ingrediente tecnológico que puede tratarse más o menos de forma objetiva, pero también cuenta con un ingrediente social por cuanto surge como una necesidad inicialmente plasmada por la ingeniería de software y en este sentido hay mucha subjetividad debido a que la ingeniería de software tiene muchos paradigmas y metodologías y, dependiendo de la que se utilice, se ve el desarrollo de sistemas de una u otra forma. Este trabajo, por tanto, aunque intenta ser lo más objetivo, no garantiza en un ciento por ciento que sea una fiel copia de lo que es la realidad en el tema de la gestión de versiones de sistemas; no obstante, intenta ser lo más cercano a lo que realmente se está viendo en las organizaciones sobre el tema.

1.5. Objetivos del trabajo

- * Dar varios puntos de vista sobre lo que son los CVS (sistemas de control de versiones).* Indagar sobre los tipos de sistemas de control de versiones.

- * Analizar el papel que cumplen los CVS en la ingeniería de software.

- * Indagar sobre los desarrollos más importantes actualmente hechos por empresas desarrolladoras de sistemas para ayudar a implementar CVS.

- * Observar las características que deben tener los buenos CVS y hacer algunas precisiones sobre el futuro de estos sistemas en el desarrollo de sistemas a largo y mediano plazo.

- * Precisar algunas conclusiones que no se encuentren en textos, sino que sean el resultado de analizar los CVS y su estado actual.

2. Contenido

Como vimos en el planteamiento del problema, los CVS son de gran importancia en la actualidad y constituyen un soporte valioso que tiene ahora la ingeniería de software para seguir fundamentando teorías que muestran a estos sistemas como sistemas estratégicos para cualquier empresa que desarrolle sistemas. Comenzaremos dando una definición de lo que se entiende por CVS y luego haremos un estudio detallado de los tipos y características de CVS, para concluir con unas prospectivas y conclusiones sobre los CVS.

2.1. CVS

Definición: CVS es básicamente un sistema de control de versiones [5][1] y podemos decir que actualmente es uno de los temas de gran interés para la ingeniería de software, debido básicamente a que si es manejado adecuadamente dicho control permitirá a los desarrolladores llevar un control estricto de sus versiones y poder volver con total seguridad a una de ellas cuando se presenten problemas posteriores en el desarrollo del sistema. La técnica más común para la implantación de un sistema de versiones es trabajar en directorios de trabajo y solo personas con permisos especiales pueden hacer envíos a dichos repositorios [6].

2.2. Tipos de sistemas de control de versiones

Lo más común en grandes empresas desarrolladoras de sistemas es que se cuente con un servidor dedicado únicamente al manejo de las versiones y dicho servidor maneja el repositorio antes mencionado. La administración del servidor debe hacerla personal altamente calificado y debe ser configurado de acuerdo con el tipo de servidor de versiones que se quiera implementar. Por ejemplo, podemos hablar de servidores de versiones concurrentes y servidores de versiones no concurrentes [6] y, sin importar el tipo de servidor de versiones que se quiera montar, se pueden establecer diferentes tipos de usuarios con respecto al servidor de versiones. Es así como encontramos usuarios administradores, usuarios anónimos e invitados [6][2].

1 Además tiene otras cosas como poder sacar instantáneas de cualquier momento de la historia, aparte de manejar unas tecnologías complejas que se describen con más detalle en [5].

2 En esta referencia se encuentra también cómo configurar estos permisos para los diversos usuarios del servidor de control de versiones.

Existe variedad de formas para implementar sistemas de control de versiones [7, 8, 9], entre las que tenemos:

- Sistema de control de versiones "clásico".
- Modelo "optimista" de control de modificaciones concurrentes.
- Clientes disponibles para muchas plataformas.
- Modelo cliente-servidor (no "desde el principio").
- Gestión conjunta de grupos de ficheros.
- Sistemas complementarios pueden soportar acceso vía web.

2.3. Los CVS y la ingeniería de software

Para comprender el papel del control de versiones en la ingeniería de software es bien importante comenzar por recordar que el objetivo principal de esta ingeniería es entregar un producto que satisfaga las necesidades del usuario, de forma eficiente y predecible [1, p. 399], y se podría pensar que este principio se ha mantenido vigente durante los últimos quince años, pues desde entonces la ingeniería de software siempre ha perseguido este fin. De otra parte, es interesante notar que la ingeniería de software ha basado el logro de este ideal utilizando dos metodologías principales: la metodología de proceso y la metodología de desarrollo, donde hoy por hoy se tiene bastante difundida la metodología de proceso RUP, que traduce proceso unificado de desarrollo[3]. Lo interesante del asunto es ver que esta metodología no solamente es compatible con una metodología de desarrollo, sino que lo es con muchas; lo que sucede es que los desarrolladores de las metodologías recomiendan que se implemente esta metodología de proceso con una metodología de desarrollo orientada a objetos; dado que las metodologías orientadas a objetos modernas usan UML como su lenguaje base, no es raro comprender que incluso se documente el proceso con UML [3, p. 25]. En este punto es importante aclarar que el proceso de desarrollo unificado es más que una metodología de desarrollo y como describen algunos autores: "Soporta las técnicas orientadas a objetos" [1, p. 400]. Pues bien, el proceso unificado de desarrollo tiene unas fases:

- Iniciación
- Elaboración
- Construcción
- Transición

En cada una de estas fases se tienen unos flujos de trabajo del proceso:

- Modelado del negocio
- Requisitos
- Análisis y diseño
- Implementación
- Pruebas
- Despliegue
- Flujos de trabajo de soporte
- Gestión del cambio y configuraciones
- Gestión del proyecto
- Entorno

La pregunta que se ha planteado la ingeniería de software es: ¿cómo hacer una verdadera gestión del cambio y configuraciones? Para responder se ha establecido una serie de pasos metodológicos y algunos de estos pasos

3 Sobre este proceso se puede encontrar bastante información en [2].

se encuentran incluso en metodologías como CMM que la incluyen en las primeras etapas para llevar los equipos de desarrollo a niveles de calidad altos [16], pero lo interesante es que desde hace muy poco se cuenta con herramientas de computación que permiten el desarrollo de esta tarea en forma efectiva. Es entonces cuando surge CVS como un sistema que se puede implementar para el control de versiones y por supuesto las grandes empresas de software han ahondado en esfuerzos tendientes a dar soluciones al problema del control de versiones e incluso, como se verá más adelante, algunas no solamente abordan el problema de las versiones, sino que quieren detallar más el proceso y piensan en el problema de las subversiones.

2.4. Sistemas para gestionar los CVS

Existen muchos desarrollos tendientes a dar soluciones a implementaciones reales de CVS; uno de ellos es "aegis" [10], el cual pone especial énfasis en la seguridad del almacén y es hoy por hoy una buena solución debido a que permite un acceso web a los administradores y usuarios del sistema. Podríamos decir que este sistema implementa la idea de "modelos de proyectos" (proyect *templates*) que no son más que plantillas en las que se consigna información sobre los diversos proyectos que son administrados por el CVS[4].

Un sistema desarrollado para manejar versiones y subversiones y que ha tenido gran aceptación en el mundo es "arch" [11], el cual posee muchas posibilidades de distribución y es bastante ligero; incluso hay quienes lo consideran el mejor sistema para el manejo de versiones. De todas formas lo único cierto es que ha tenido gran aceptación mundial y un crecimiento grande del número de empresas y entidades que lo implementan.

Existe un sistema llamado "Vesta" [12], el cual permite el control de versiones y adicionalmente está concebido con énfasis en el rendimiento y ha sido pensado para grandes sistemas; lo malo es que hasta el momento ha sido difícil de portar, por cuanto actualmente funciona solamente sobre Linux.

Para el caso de Microsoft tenemos que esta gran empresa de desarrollo de soluciones informáticas ha pensado en dar soluciones específicas para desarrolladores pequeños y grandes. Para las empresas de desarrollo pequeñas o desarrolladores independientes la solución es "Visual Source Safe 2005" y para desarrollos complejos la solución se basa en "Visual Studio 2005 Team System: Enterprise Class Source Control and Work Item Tracking" [13]. Entre las cosas más atractivas de las soluciones ofrecidas por Microsoft, podemos citar el hecho que dichas herramientas ofrecidas por esta empresa para el control de versiones cubren aspectos como el modelado, la documentación, las pruebas, la administración del software, y se podría decir que son herramientas que satisfacen las necesidades completas para la administración del ciclo de vida del software.

2.5. Otros sistemas que apoyan a los CVS

Algunos desarrollos de software han hecho que los CVS indirectamente cojan fuerza y sean día por día una realidad en la ingenie-

4 Jesús M. González Barahona, profesor de la Universidad de Juan Rey en España, dice que "aegis" tiene una funcionalidad similar a los CVS. Se puede contactarlo en jgb@computer.org; jgb@gsyc.escet.urjc.es

ría de software. Tenemos por ejemplo muchas herramientas tipo GNU[5], que ayudan a definir y catalogar cadenas de texto que van a ser usadas por los programas[6], y otras que ayudan a automatizar la gestión de cambios de idioma [14][7].

2.6. Características que debe tener un buen control de versiones

Antes de mostrar algunas de las características más importantes que debe tener un buen control de versiones, es necesario pensar en algunas cuestiones preliminares que ayuden a comprender el tema y la importancia del control de versiones.

Cuando se habla de control de versiones, se piensa inmediatamente en el código fuente de las aplicaciones, pero es necesario comprender que el control de versiones no solamente debe abarcar este aspecto sino que debe cubrir otros como los requisitos, los diagramas, la documentación. Establecer cuándo se llega a una versión es algo importante y debe obedecer a políticas institucionales de desarrollo de versiones. No se deben establecer versiones definitivas sin antes haber pasado dichos productos por otro tipo de versiones. En este sentido, un buen mecanismo para establecer versiones es definir tipos de versiones, por ejemplo: alfa, beta y definitivas.

Veamos entonces algunas de las características que debe tener un buen control de versiones:

Un sistema de control de versiones debe, primero que todo, darle al desarrollador la posibilidad de establecer el momento en que desea indicarle al sistema que tiene una versión. Obviamente este momento tiene que estar acompañado de un almacenamiento confiable de toda la información del proyecto en el momento de tomar la decisión de establecer la fijación de una versión.

De otra parte, de la anterior característica surge otra bien importante: la posibilidad que debe tener el desarrollador de sistemas de conocer cuáles son las diferencias entre lo que se tiene actualmente y la última versión, con el fin de saber si es el momento preciso para una nueva versión, o si todavía los esfuerzos hechos no ameritan el lanzamiento de una nueva versión.

Los sistemas de control de versiones deben soportar varios programadores trabajando al mismo tiempo, algo similar a lo que pasaba cuando comenzaron los primeros entornos de desarrollo para trabajar en aplicaciones web. Recordemos por ejemplo lo que hizo Microsoft con su herramienta inicial Microsoft Visual Interdev 6.0 en la que se trabajaban simultáneamente los conceptos de servidor de producción (que procesaba las páginas), servidor maestro (que procesaba las copias principales) y servidor local (que se encontraba en una estación de trabajo y proporcionaba funcionalidades necesarias para probar todos los tipos de elementos web antes de extenderlos a la aplicación web maestra) [4, p. 476]. Pues bien: en los sistemas de control de versiones se mejoran es-

5 Para saber más sobre GNU, se puede entrar a http://www.gnu.org

6 Estos programas ayudan a catalogar programas y algunos autores los denominan "catálogo de mensajes".

7 Es importante notar que para automatizar la gestión de cambio de idioma se deben establecer primero reglas simples para especificar las traducciones que se deben hacer, tal como lo menciona Jesús M. González Barahona.

tos conceptos y se permiten seguridades más grandes y controles no solamente a nivel de código fuente sino de auditoría para saber en determinado momento cuál desarrollador hizo un cambio que dañó el desarrollo del sistema, para que sea almacenado por medio de registros recuperables que permitan con un alto grado de seguridad recuperar la versión con toda su funcionalidad.

Los buenos sistemas de control de versiones deben permitir al equipo desarrollador recuperar versiones previas en el desarrollo del ciclo de vida del software, que muestren no solamente las líneas que han cambiado, sino la documentación que ha sido agregada o eliminada, entendiendo por documentación los diagramas, los diccionarios, la documentación del código fuente, tal como, por ejemplo, lo implementa Visual Source Safe [17].

Los sistemas de control de versiones deben tener capacidades de compartir y enlazar archivos de desarrollo para promover la reutilización de código y componentes entre proyectos y simplificar el mantenimiento de código al preparar los cambios entre todos los archivos compartidos y enlazados cuando un archivo sea actualizado, lo cual, por ejemplo, es algo que también implementa Visual Source Save [17].

Las características de desarrollo en paralelo, como el *branching*, permiten que los equipos separen el proceso de desarrollo en proyectos y archivos paralelos, creando copias idénticas que heredan toda la documentación de versiones, pero a las que se les puede dar seguimiento como proyectos nuevos e individuales [17]. Esto obviamente debe estar soportado por una alta tecnología que permita reconciliar diferencias entre diferentes versiones del mismo archivo y ver cómo se comportaría una determinada versión al ser incluida dentro del proyecto, pero todo esto bajo la premisa de evitar al máximo la pérdida potencial de cambios valiosos.

Al igual que con los usuarios de cualquier otro sistema, los sistemas de control de versiones deben permitir manejar niveles de usuarios; aunque globalmente todos sean desarrolladores de software y de sistemas, es bueno que el sistema de control de versiones permita la gestión de usuarios en el equipo de desarrollo basado en políticas de seguridad y niveles de permisos [7, 8].

2.7. El futuro de los CVS

Como sucede con casi todas las cosas del mundo, los desarrollos tecnológicos también están en constante cambio y evolución; pues bien, los CVS, que fueron concebidos para manejar versiones, pueden ser desplazados por los sistemas que manejan subversiones, los cuales, aparte de ser netamente web, permiten gestionar directorios renombrados y tienen algoritmos bastante eficientes para hacer búsquedas y para gestionar subversiones [15].

3. Conclusiones sobre los CVS

Es importante notar que los CVS han evolucionado a sistemas que permiten un alto grado de confiabilidad para la gestión de versiones en el ciclo de vida de sistemas y se enfocan principalmente a permitir el trabajo de múltiples usuarios que físicamente se pueden encontrar en cualquier parte del mundo. Podemos decir entonces que, aunque los CVS que existen actualmente funcionan preferencialmente en entornos LAN, también soportan WAN de forma cada vez más robusta.

Se puede ver claramente que de todas formas existe responsabilidad grande delegada al administrador del CVS, debido a que de los permisos que este conceda a los usuarios y de la buena gestión que haga del sistema dependerá en buena medida el éxito del manejo de versiones en un proyecto de desarrollo. Esto obviamente tiene una limitación y es que el administrador de control de versiones debe ser una persona con alto grado de conocimientos de programación, y buena parte de la responsabilidad del control de versiones recae sobre el mismo.

Cuando surgen problemas en el desarrollo del sistema y se toma la decisión de volver a una versión previamente almacenada, se puede fácilmente volver con las herramientas actuales, pero si por desgracia se quiere volver a un tiempo en el ciclo de desarrollo del sistema que no haya sido previamente almacenado y establecido en el sistema, se tienen muchas limitaciones en todos los sistemas CVS actuales.

También podemos decir que, aunque los CVS actuales permiten combinar archivos diferentes que correspondan a versiones diferentes, existe todavía un alto grado de posibilidad que, al hacerlo, un archivo pueda estropear indirectamente otras cosas, pero que no sea tan evidente. En estos casos decimos que, para reconciliar de una mejor manera los problemas al mezclar archivos, se deben hacer simulaciones de cómo sería la integración del sistema, pero dichas simulaciones todavía no son posibles con los CVS actuales.

Otra cuestión bien relacionada con la conclusión anterior es que al combinar diversas versiones se toman decisiones basadas en la experiencia, el buen conocimiento de programación y otras cuestiones que de ninguna manera pueden ser remplazadas por sistemas actuales. Se dice entonces que técnicas como la inteligencia artificial todavía no se han integrado de forma plena a los CVS actuales.

Se puede pensar que los actuales CVS están diseñados para gestionar versiones de desarrollo de software en lenguajes específicos; por ejemplo, los sistemas creados por Microsoft son para utilizar en equipos de desarrollo que usen herramientas de Microsoft; de la misma forma otros sistemas en software amparado por GNU siguen siendo específicos para ciertos lenguajes de programación, y esto obviamente lleva a pensar que se está muy lejos de construir sistemas de control de versiones que soporten desarrollos de sistemas basados en múltiples lenguajes.

Bibliografía

[1] Booch, Grady, Rumbaugh, James, Jacobson, Ivar. *El lenguaje unificado de modelado*. Addison Wesley. 1999.

[2] Booch, Grady, Rumbaugh, James, Jacobson, Ivar. *El lenguaje unificado de modelado*. Addison Wesley. 2002.

[3] Booch, Grady, Rumbaugh, James, Jacobson, Ivar. *The unified software development process*. Addison Wesley Object Technology Series. 2003.

[4] Press Microsoft. *Microsoft Visual Interdev 6.0. Manual del programador*. McGraw-Hill. 1999.

Infografía

[5] http://gsyc.escet.urjc.es/pub/actvidades/linuxprog/prog-html.tar.gz Copias: http://gsyc.escet.urjc.es/pub/actividades/linuxprog/prog-dvi.tar.gz http://gsyc.escet.urjc.es/pub/actividades/linuxprog/prog.ps.gz; 31 de mayo de 2001.

[6] http://qref.sourceforge.net/Debian/reference/reference.es.html#contents; 14 de enero de 2005.

[7] http://www.cvshome.org; 30 de octubre de 2003.

[8] http://www.gnu.org/software/cvs/cvs.html; 27 de septiembre de 2004.

[9] http://cvsbook.red-bean.com; 3 de enero de 2005.

[10] http://aegis.sourceforge.net/index.html; 13 de agosto de 2005.

[11] http://regexps.com/arch.html; 14 de enero de 2005.

[12] http://www.vestasys.org; 17 de diciembre de 2005.

[13] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvsent/html/vsts-arch.asp; 5 de febrero de 2005.

[14] http://www.gnu.org/software/gettext/gettext.html; 10 de septiembre de 2004.

[15] http://subversion.tigris.org; 17 de diciembre de 2005.

[16] http://www.acis.org.co/index.php?id=108; 18 de octubre de 2005.

[17] http://www.microsoft.com/latam/ssafe/producto/resumen.asp; 29 de diciembre de 2004.