ANÁLISIS Y EVALUACIÓN DE HERRAMIENTAS DE CONTROL DE VERSIONES EN PROYECTOS SOFTWARE

Patricia Miravet

CTIC – Centro Tecnológico

Carlos Alba

ArcelorMittal Global R&D Asturias

Vicente Rodríguez, Luis Fernández

Universidad de Oviedo, Área de Proyectos de Ingeniería

Abstract

One of the major problems to face in software projects is the version management, not only when the project is developed by different people, but also in the organization and structure at individual level. To solve this problem, version management tools are used, which facilitate mainly the storing of the items to be managed, the restoring of them and the historical record and identification of every modification carried out in the different versions of the project code.

This work presents a study on the different existing tools used for version management in software projects existing nowadays in the market, analyzing their features and functionalities in order to establish assessment and benchmarking criteria. Starting from these assessments, the key features to include in the development of the ideal versioning tool will be determined, satisfying the real needs of the project teams.

Included in the results of this work, there are statistical data that reflect the trends and new strategies in this type of tools in the current market, as support in the management of software projects.

Keywords*: software project, version control, benchmarking*

Resumen

Uno de los mayores problemas a afrontar en los proyectos software es el control entre versiones, no solo cuando éste es desarrollado por distintas personas, sino también en la organización y estructuración a nivel individual. Para solventar dicho problema, se recurre a las llamadas herramientas de control de versiones, las cuales facilitan principalmente el almacenamiento de los elementos a gestionar, la recuperación de cada uno de ellos y el registro histórico e identificación de cada una de las modificaciones realizadas en las sucesivas versiones del código del proyecto.

Este trabajo presenta un estudio de las diferentes herramientas destinadas a la gestión de versiones en proyectos software existentes en el mercado a día de hoy, analizando sus características y funcionalidades para establecer unos criterios de valoración. A partir de dichas evaluaciones se determinarán las características esenciales para la creación de una herramienta idónea para satisfacer las necesidades reales de los equipos de proyecto.

Se incluyen en los resultados de este estudio datos estadísticos que reflejan la tendencia y las nuevas estrategias de este tipo de herramientas en el mercado actual, como ayuda en la gestión de proyectos software.

Palabras clave: *proyecto software, control de versiones, benchmarking*

1. Introducción

Tradicionalmente el desarrollo de software se relegaba a un pequeño grupo de programadores, que con una filosofía común realizaban su trabajo de manera colaborativa gestionando las distintas versiones de forma manual. Esta gestión era ineficiente, con grandes pérdidas de tiempo intentando encontrar la última versión de un archivo concreto, o en saber si un fichero ha sido modificado, cuándo y por quién.

El desarrollo de software es en la actualidad algo mucho más amplio, generalmente muy dinámico y los productos generados altamente susceptibles al cambio. Ni se reduce a una labor individual, ni a un pequeño grupo de desarrolladores, sino que son muchos los desarrolladores que necesitan acceder en paralelo al mismo código fuente y realizar sus propias modificaciones. Intentar coordinar manualmente las diferentes modificaciones concurrentes sobre el mismo elemento mediante convenciones comunes es una labor tediosa e incluso puede llegar a ser imposible en cuanto el grupo de trabajo aumenta y se dispersa. Por ello se hace imprescindible el uso de herramientas que potencien la comunicación y la coordinación entre todos ellos de manera automática y de forma más eficiente, los sistemas de control de versiones.

Los sistemas de control de versiones se encargan de almacenar el historial de las modificaciones que van sufriendo los diferentes archivos de un proyecto, es decir, de gestionar los sucesivos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo manteniendo de manera estructurada los avances, retrocesos y modificaciones del software mientras está siendo desarrollado.

Las ventajas de utilizar un sistema de control de versiones son innegables, no solo son eficaces en la gestión del trabajo concurrente de varios desarrolladores permitiendo manejar las diferentes versiones del proyecto, sino que permiten seguir la evolución del mismo. Cada ínfimo cambio estará registrado en el sistema con todos los datos que se puedan llegar a necesitar (qué, quién, por qué, cuando, etc.). Favorecen la colaboración entre los integrantes del equipo facilitando la integración de sus versiones, encontrando cambios potencialmente incompatibles y ayudando a resolver tales conflictos. Sin dejar de lado la posibilidad de solucionar los errores introducidos con mayor facilidad volviendo a una versión anterior con solo elegir la deseada.

No obstante lo anterior y contando con todas las ventajas que ofrece un sistema de control de versiones no se debe olvidar que por el momento ninguna herramienta es capaz de sustituir la comunicación entre los desarrolladores, y en último caso, de existir cambios incompatibles sobre la misma información, será necesaria la intervención humana para tomar la decisión definitiva.

2. Objeto

En el proceso de desarrollo de software los cambios son inevitables debido a que los requerimientos de los sistemas siempre cambian durante su desarrollo y uso. Estos cambios contribuyen a aumentar el grado de confusión de los desarrolladores cuando no han sido

correctamente analizados y/o registrados, no han sido comunicados apropiadamente a las personas implicadas o no han sido estrechamente controlados (Pressman & Ince, 1982).

El uso de herramientas de control de versionado es esencial para una gestión del cambio eficaz. En la actualidad, la gestión de versiones se apoya siempre en herramientas CASE que administran el almacenamiento de cada versión del sistema y controlan el acceso a los diferentes componentes del mismo. A pesar de que las herramientas difieren obviamente en funcionalidades de interfaz, la base de todas estas herramientas de soporte es el control de la gestión de versiones.

Desde principios de los años 70 se han producido un gran número de diferentes herramientas con este propósito y el objeto de este trabajo es realizar un análisis de las más significativas y una evaluación de sus características más destacables. Para ello se ha realizado una comparativa en la que se ponen en común las distintas peculiaridades de cada una de ellas, detallándola en diferentes tablas que permitan comprobar los pros y contras de cada herramienta.

A partir de aquí se definen las características esenciales para la creación de una herramienta de gestión de versiones idónea capaz de satisfacer las necesidades reales de los desarrolladores de software.

Además se realiza un estudio del mercado actual para marcar la tendencia y las nuevas estrategias seguidas por los creadores de este tipo de herramientas aportando datos estadísticos que corroboran las conclusiones obtenidas.

Se complementa este trabajo con una pequeña metodología, genérica, de selección aplicable por cualquier tipo de organización.

3. Estado del Arte

El problema de la gestión de las diferentes versiones de un proyecto software se remonta a principios de la década de los 70, cuando los desarrolladores se dieron cuenta de que seguir gestionando las diferentes versiones a mano era algo insostenible.

Por ello, en 1972 aparece *SCCS* (Source Code Control System) (Rochkind, 1975) que únicamente llevaba el control de archivos individuales y almacenamiento local de las diferentes versiones. Aunque su funcionamiento era sumamente básico tenía un diseño bien planteado y de hecho su formato interno sigue usándose por *TeamWare* (Sun Microsystems, 2001) y *BitKeeper* (BitMover, 2010).

Tras varios años de utilización de este sistema, en 1982 nace *RCS* (W. F. Tichy, 1982), como un intento de competencia al monopolio de *SCCS* y aunque realmente no supuso ningún gran avance tecnológico todavía se puede encontrar en algunas distribuciones de Linux debido a su facilidad de uso. Las carencias de estos sistemas son visibles a simple vista: únicamente dan soporte para texto, no existe un repositorio central y la nomenclatura del versionado es un tanto hostil. Debido a todo esto surge la segunda generación de sistemas de control de versiones que incorpora mejoras tan importantes como el control de árboles de archivos o el almacenamiento de las revisiones de manera centralizada. Es la época cliente–servidor, sistemas que permiten gestionar múltiples versiones desarrolladas de forma concurrente en diferentes máquinas y almacenadas en un servidor central.

Esta segunda generación comienza con la aparición de *CVS* en 1990 (Cederqvist & Pesch, 1993) y marca el comienzo de los sistemas de control de versiones modernos. *CVS* es capaz de gestionar versiones de forma eficiente e incluso soporta ramificación y fusiones (branching y merging). Aún así seguía teniendo múltiples inconvenientes entre ellos la necesidad de bloquear todo el repositorio para la realización de muchas de sus operaciones,

con la consiguiente pérdida de tiempo por parte de todos los desarrolladores (Berliner, 1990).

Otros sistemas de control de versiones nacieron en esta época con más o menos éxito. En 1992 aparece *ClearCase* (Galfullin & Novichkov, 2000), un sistema adelantado a su época, con una potencia extraordinaria y ciertas características inigualables en cuanto a ramificación y fusiones pero que sin una evolución continuada y una complejidad desmesurada cayó en el olvido. En 1995, *Perforce* entra en escena (Bjorkholm, 1997), no llegaba a la inmensa funcionalidad del anterior pero sin ninguna duda ganaba en cuanto a precio, rendimiento y facilidad de uso. Para finalizar con esta generación en el mismo año aparece *Microsoft Visual SourceSafe* (VSS) (Serban, 2007) con la gran ventaja de proporcionar un interfaz gráfico que facilitaba enormemente la interacción con el sistema aunque era bastante lento y limitado.

En 2000, *Subversion* (Collins-sussman, Fitzpatrick, & Pilato, 2008) marca realmente un punto y aparte con todos los demás sistemas, provocando el inicio del uso masivo de sistemas de control de versiones y el comienzo de la tercera generación de este tipo de sistemas. *Subversion* intenta solventar los inconvenientes de los sistemas de la segunda generación con la inclusión del almacenamiento centralizado de las diferentes revisiones, el control de árboles de archivos de manera eficiente y el manejo completo de operaciones complejas con los mismos.

Aunque otros sistemas intentaron obtener su cuota de mercado en esta época, muy pocos lo consiguieron. Cabe resaltar *Accurev* (Accurev Inc., 2010) que aunque debido a la supremacía total de *SVN* no tuvo la expansión esperada consiguió que muchas empresas migraran de sistemas más antiguos como *CVS* a sistemas centralizados.

La cuarta generación da paso a una nueva filosofía de sistemas de control de versiones. Desaparece la idea de tener todas las versiones centralizadas en un único nodo y aparece el término descentralizado como modo de almacenaje en diferentes nodos. El sistema que marca el inicio de este gran cambio de visión sobre los sistemas de control de versiones fue *GNU Arch* en 2001 (Moffitt, 2004), si bien el manejo de los flujos de trabajo era aún deficiente. Esta etapa llega a su plenitud con la aparición de sistemas como *Bazaar* (Bazaar Developers, 2011), *Git* (Loeliger, 2009) o *Mercurial* (O'Sullivan, 2009), en los que cada usuario tiene su propio repositorio del cual es administrador y tiene el control para tomar cualquier decisión, es decir, permite el manejo de múltiples flujos de trabajo simultáneos, y lo que es más importante no rompe con todo, sino que también permite el trabajo centralizado.

Otro sistema que apareció a la par que estos tres anteriores fue *Darcs* (Roundy, 2005), un sistema que ha seguido evolucionando a lo largo de los años y alcanzó la versión 2.5 en Noviembre de 2010. Aunque su evolución y la constante publicación de nuevas versiones son dos grandes ventajas respecto a otros sistemas, sus grandes limitaciones en cuanto a la dificultad de seguir la historia de un archivo concreto y su bajo rendimiento hacen que su uso no esté muy extendido.

En 2006 nace *Plastic SCM* (Software, 2011) que proporciona un entorno de trabajo fundamentalmente gráfico, como multitud de visualizaciones del árbol de la historia que facilitan enormemente el trabajo al desarrollador. Además soporta el desarrollo desconectado. En este año también aparece *Fossil* (Schimpf, 2010) incluyendo una wiki, un sistema de tickets, un servidor Web, y un blog.

En 2008 sale al mercado *IBM Rational Team Concert* (Team Server, 2008) como una solución completa de colaboración para equipos de desarrollo de todos los tamaños, que permite la gestión de trabajo, información y comportamiento de forma unificada a lo largo del ciclo de vida de los proyectos software.

4. Metodología

El estado del arte anterior muestra un resumen de las herramientas de control de versiones en proyectos software existentes en el mercado que supusieron un cambio de filosofía o una mejora considerable respecto a las demás.

En este trabajo se han elegido catorce herramientas y se ha realizado una comparación estudiando diferentes características consideradas de gran importancia por los desarrolladores, según lo leído en la bibliografía. Inicialmente se ha tenido en cuenta información general de las herramientas, clasificándolas en productos comerciales o de software libre, comparando precios y licencias. Se han tenido en cuenta las fechas de la primera versión liberada como la última existente para medir la antigüedad, que en la mayoría de los casos demuestra su solidez, y el ritmo de actualización como medida de implicación de los desarrolladores en el proceso de mejora de la misma. Por otra parte se ha contemplado el estado de desarrollo del sistema comprobando si actualmente se ha detenido, si éste sigue activo o si únicamente se realizan tareas de mantenimiento y resolución de incidencias. Otro apartado general analizado es la información técnica disponible sobre la herramienta, en concreto el tipo de repositorio que maneja cada una, ya sea centralizado o distribuido, y el modelo de concurrencia utilizado, para contribuir a formar una idea general acerca de la tendencia general del mercado.

A continuación se analiza la funcionalidad específica de cada herramienta incluyendo una exhaustiva serie de opciones y comandos, comparando cuales son los más generalizados y cuales han conseguido que algunas de las herramientas más antiguas caigan en el desuso.

Finalmente se realiza un resumen de las interfaces con las que cuenta cada una de las herramientas, tanto interfaces Web como de escritorio y se incluyen también los plugins disponibles para su integración con los entornos de desarrollo más utilizados por los programadores hoy en día.

5. Resultados

5.1. Información General

La comparativa realizada sobre las características generales de las herramientas analizadas se muestra en la **Tabla 1**.

Tabla 1: Tabla comparativa—Información general

Herramienta	Precio / Licencia	Primera versión	Versión Actual	Estado
AccuRev	$1,495 por licencia	2002	Noviembre, 2010 (v.4.9)	Activo
Bazaar	GPL	2007	Febrero, 2011 (v.2.3.0)	Activo
IBM Rational ClearCase	$4,125 por licencia	1992	Diciembre, 2009 (v.7.1.1)	Activo
CVS	GPL	1990	Mayo, 2008 (v.1.11.23)	Mantenimiento
Darcs	GPL	2002	Febrero, 2011 (v. 2.5.1)	Activo
Fossil	2-clause BSD license	2006	Marzo, 2011 (v.4cd9309c50)	Activo
Git	GPLv2	2005	Febrero, 2011 (v.1.7.4.1)	Activo
GNU Arch	GPL	2001	Julio, 2006 (v.1.3.5)	Mantenimiento
Mercurial	GPL	2005	Marzo, 2011 (v.1.8)	Activo

Perforce	$900 por licencia	1996	Febrero, 2010 (v.2010.2)	Activo
Plastic SCM	$595 por licencia	2006	Noviembre, 2010 (v.3.0.10)	Activo
IBM Rational Team Concert	$168 por licencia	2008	Noviembre, 2010 (v.3.0)	Activo
Subversion (SVN)	Apache License	2000	Noviembre, 2010 (v.1.6.15)	Activo
Microsoft Visual SourceSafe (VSS)	$500 por licencia	1994	Octubre, 2005 (v.8.0.50727.42)	Mantenimiento

A la hora de elegir una herramienta de gestión de versiones es importante tener en cuenta ya no solo el precio a pagar por licencia sino también la asidua actualización de versiones llevada a cabo por el fabricante. Respecto al precio o licencia asociada, se observa que las herramientas de software libre optan generalmente por una licencia GNU General Public License que permite la libre distribución, modificación y uso del software. Las herramientas comerciales presentan gran variabilidad en el precio, desde $168 de *IBM Rational Team Concert* a los $4,125 de *Perforce* por usuario, por lo que las características que ofrezcan deben ser destacablemente superiores a las de las herramientas de software libre para que su compra resulte rentable.

Excluyendo sistemas como *CVS, GNU Arch o Microsoft Visual SourceSafe* que se encuentran en fase de mantenimiento, el resto de herramientas han liberado su versión más reciente en el último año y medio. Más estrictamente el 81% de las herramientas estudiadas en este artículo ha sacado una nueva versión en los últimos cuatro meses y un 36% la ha sacado en el último mes, lo que da una idea del trabajo constante llevado a cabo, continuado y reciente.

5.2. Información Técnica

La **Tabla 2** presenta una comparativa respecto a parámetros técnicos, que se describen brevemente a continuación para facilitar su lectura:

- Tipo de repositorio: centralizado si existe un único repositorio central para todo el proyecto en un servidor dedicado o distribuido si cada usuario tiene su propio repositorio del cual es administrador
- Modelo de concurrencia: lock-modify-unlock (LMU) si el sistema sólo permite a una persona modificar un archivo al mismo tiempo y se utiliza la técnica del bloqueo para conseguirlo o copy-modify-merge (CMM) si el sistema permite a múltiples usuarios trabajar de manera simultánea sobre las copias locales que tienen del repositorio
- Modelo de almacenamiento: historia del cambio (changeset) si almacena únicamente el conjunto de los nuevos cambios realizados o snapshot si almacena una instantánea del árbol de directorios antes y después del cambio
- Alcance del cambio: árbol o ficheros individuales
- Identificación de las revisiones: números, pseudoaleatorio o funciones hash criptográficas SHA-1 hashes

Tabla 2: Tabla comparativa—Información técnica

Herramienta	Tipo de repositorio	Modelo de concurrencia	Historia de cambios	Alcance del cambio	Identificación de revisiones
AccuRev	Centralizado	LMU / CMM	Changeset	Árbol	Números
Bazaar	Distribuido	CMM	Snapshot	Árbol	Pseudoaleatorio

ClearCase	Centralizado	LMU / CMM	Changeset	Fichero	Números
CVS	Centralizado	CMM	Changeset	Fichero	Números
Darcs	Distribuido	CMM	Patch	Árbol	Números
Fossil	Distribuido	CMM	Snapshot	Árbol	SHA-1 hashes
Git	Distribuido	CMM	Snapshot	Árbol	SHA-1 hashes
GNU Arch	Distribuido	CMM	Changeset	Árbol	Números
Mercurial	Distribuido	CMM	Changeset	Árbol	SHA-1 hashes
Perforce	Centralizado	LMU / CMM	Changeset	Árbol	Números
Plastic SCM	Centralizado	LMU / CMM	Changeset	Árbol	Números
IBM Rational Team Concert	Distribuido	LMU / CMM	Changeset	Árbol	Números
Subversion (SVN)	Centralizado	LMU / CMM	Changeset / SnapShot	Árbol	Números
Visual SourceSafe (VSS)	Centralizado	LMU / CMM	Snapshot	Fichero	Números

Esta tabla pone de manifiesto la clara diferencia entre las herramientas más novedosas y las más antiguas, el tipo de repositorio. Todas las herramientas existentes desde 2006 se han decantado por un modelo distribuido y por un modelo de concurrencia centrado en CMM que evita los bloqueos innecesarios.

En cuanto a la manera de almacenar el historial de cambios del repositorio existen dos variantes mayoritarias y claramente diferenciadas, snapshot y changeset. Changeset realiza ramificación y fusiones de forma más rápida al no tener que comparar instantáneas de árboles de directorios completos sino sólo los cambios realizados entre las diferentes versiones. Además el espacio en disco necesario para almacenar estos cambios es menor.

El alcance del cambio tiene una tendencia generalizada a ser a nivel de árbol en todas las herramientas estudiadas a partir de 1994 y la identificación de las versiones en las herramientas más novedosas como *Git, Mercurial* o *Fossil* ya se realiza con SHA-1 hashes.

5.3. Opciones

El análisis de las diferentes opciones permitidas en cada una de las herramientas se muestra en la **Tabla 3**. Para cada una de ellas se ha rellenado la casilla correspondiente con una letra (S: Sí, N: No, P: Parcial) según corresponda.

En concreto se han evaluado las siguientes características:

- Commits atómicos: garantiza que todos los cambios realizados han sido fusionados o que ninguno lo ha sido.
- Renombrado de ficheros y directorios: describe si el sistema es capaz de mantener la historia de cambios de ficheros o directorios que han sido copiados o movidos.
- Fusión de ficheros renombrados: describe si el sistema es capaz de realizar fusiones de ficheros o directorios que han sido renombrados sin perder por ello la historia de cambios.
- Trazas de fusiones realizadas: describe si la herramienta es capaz de mantener actualizada la historia de cambios a raíz de las fusiones realizadas.

- Réplica de repositorio: indica si el sistema es capaz de realizar una copia íntegra del repositorio.
- Permisos del repositorio: indica si es posible definir diferentes permisos para los ficheros o directorios existentes en el repositorio.
- Traza de la historia del fichero o directorio: indica si el sistema es capaz de llevar la historia completa de cambios de un determinado fichero o directorio.
- Tags: permite la creación de etiquetas para identificar las diferentes revisiones.
- Internacionalización: indica si el sistema tiene soporte para diferentes lenguajes.
- Unicode: indica si el sistema puede realizar operaciones con ficheros que usen diferente codificación de caracteres.

Tabla 3: Tabla comparativa—Opciones

Herramienta	Commits atómicos	Renombrado de ficheros y directorios	Fusión de ficheros renombrados	Trazas de fusiones realizadas	Replica de repositorio	Permisos del repositorio	Traza de la historia del fichero o directorio	Tags	Internacionalización	Unicode
AccuRev	S	S	S	S	S	S	S	N/A	S	N
Bazaar	S	S	S	S	S	S	S	S	S	S
ClearCase	P	S	S	S	N	S	S	S	S	S
CVS	N	N	N	S	N	P	S	S	N	N
Darcs	S	S	S	N	S	N	S	S	N	S
Fossil	S	S	S	N	S	S	S	S	S	S
Git	S	P	S	S	S	S	S	S	P	P
GNU Arch	S	S	S	S	S	S	N	S	N	N
Mercurial	S	S	S	S	S	S	S	S	S	N
Perforce	S	S	N	S	S	S	S	S	S	S
Plastic SCM	S	S	S	S	S	S	S	S	N	S
IBM Rational Team Concert	S	S	S	S	N	S	P	S	S	S
Subversion (SVN)	S	S	N	S	N	N	S	P	S	S
Visual SourceSafe (VSS)	N	N	N	N	N	S	N	S	S	S

Este apartado contiene las características más requeridas por los desarrolladores de software en los sistemas de gestión de versiones. Se comprueba que sistemas como *Bazaar, Git, Fossil* o *Mercurial*, los más novedosos en el mercado, poseen más del 90% de las características arriba mencionadas.

Una de las grandes desventajas que presentaban muchos sistemas de control de versiones antiguos y que hoy en día todavía poseen herramientas como CVS o VSS es la imposibilidad de renombrar ficheros o copiarlos a otras rutas manteniendo su historial de versiones. Como se puede comprobar en la tabla anterior, aunque en sus primeras versiones no fuera así, a fecha de hoy casi todas las herramientas han suplido ya esta carencia y actualmente sí lo permiten.

Otra característica demandada por los desarrolladores de software es la posibilidad de utilizar el estándar de codificación de caracteres UNICODE, y habiendo comprobado las últimas versiones de las herramientas sometidas a estudio se concluye que el 71% de las mismas ya cumplen con esta característica.

5.4. Comandos Básicos

Se muestra en este apartado una comparativa sobre los comandos básicos soportados por cada una de las herramientas, explicados a continuación y disponibles en la **Tabla 4**:

- Init: creación de un nuevo repositorio.
- Clone: creación de una copia idéntica de un repositorio ya existente.
- Pull: descarga de los cambios de un repositorio remoto a uno local.
- Push: subida de los cambios introducidos en un repositorio local a uno remoto.
- Branch: creación de ramas locales inexistentes en el repositorio remoto.
- Checkout: creación de una copia local del repositorio.
- Update: actualización del repositorio local a partir del repositorio remoto.
- Lock: bloquear ficheros del repositorio para que otro usuario no pueda modificarlos.
- Add: marcar los ficheros que deberán ser añadidos al repositorio en el siguiente commit.
- Remove: marca los ficheros a borrar del repositorio en el siguiente commit.
- Move: marca los ficheros a mover a otro directorio del repositorio en el siguiente commit.
- Copy: marca los ficheros que deberán ser copiados en el siguiente commit.
- Merge: aplica las diferencias entre dos ficheros.
- Commit: guarda los cambios en el repositorio.
- Revert: restaura la copia local a partir de la del repositorio

Tabla 4: Tabla comparativa—Comandos básicos

Herramienta	init	clone	pull	push	branch	checkout	update	lock	add	remove	move	copy	merge	commit	revert
AccuRev	S	N	N	S	S	S	S	S	S	S	S	N	S	S	S
Bazaar	S	S	S	S	S	S	S	N	S	S	S	N	S	S	S
ClearCase	S	N	N	N	N	S	S	S	S	S	S	N	S	S	S
CVS	S	N	N	N	N	S	S	S	S	S	N	N	S	S	S
Darcs	S	S	S	S	N	S	S	S	S	S	S	N	S	S	S
Fossil	S	S	S	S	S	S	S	N	S	S	S	N	S	S	S

Git	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
GNU Arch	S	N	S	N	S	S	S	N	S	S	S	S	S	S	S
Mercurial	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
Perforce	S	N	N	N	N	S	S	S	S	S	S	S	S	S	S
Plastic SCM	S	N	N	N	N	S	S	N	S	S	S	S	S	S	S
IBM Rational Team Concert	S	N	N	N	S	S	S	S	S	S	S	S	S	S	S
Subversion (SVN)	S	S	S	S	N	S	S	S	S	S	S	S	S	S	N
Visual SourceSafe (VSS)	S	S	S	S	N	S	S	S	S	S	S	S	S	S	S

De la tabla comparativa mostrada anteriormente se pueden sacar varias conclusiones. Inicialmente comandos básicos como init, checkout, commit o update son soportados por el 100% de las aplicaciones, comandos relacionados con el renombrado de archivos o su copia se reducen a herramientas más actuales que permiten estas acciones y finalmente otros comandos más complejos como pull, push o lock reducen su aparición a las herramientas de control de versiones distribuidas.

Por otra parte cabe destacar la escasa diferencia entre herramientas comerciales y herramientas de código libre, proporcionando tanto unas como otras el mismo número de operaciones.

5.5. Interfaces de Usuario

La **Tabla 5** muestra el resumen de interfaces de usuario disponibles para cada una de las herramientas estudiadas; interfaces Web, interfaces gráficas (GUI) y plugins para ser integradas con diferentes entornos de desarrollo (IDE).

Tabla 5: Tabla comparativa—Interfaces de usuario

Herramienta	Interfaz Web	GUI	Integración / Plug-ins IDE
AccuRev	SI	Windows, Linux, Unix, Mac OS X, BeOS	IntelliJ IDEA Eclipse, Visual Studio
Bazaar	Permite la utilización de un servidor web.	Olive, bzr-gtk (GTK+), Bazaar Explorer (Qt), QBzr (Qt), TortoiseBzr (Windows)	Eclipse (BzrEclipse, QBzrEclipse), Visual Studio (bzr-visualstudio), TextMate (TextMateBundle), Komodo IDE
ClearCase	Incluido, Clearcase Web Interface	MS Windows, Unix, clientes TSO para z/OS.	Emacs, Eclipse (Eclipse-CCase), Visual Studio, Kdevelop, IntelliJ IDEA
CVS	cvsweb, ViewVC, codeBeamer	Windows, Mac OS X, OS/400, GTK, Qt	SCCI, Bugzilla, Build
Darcs	Incluido darcs.cgi, darcsweb, TortoiseDarcs	Windows, Mac OS X	Eclipse (eclipsedarcs), Emacs (vc-darcs.el)
Fossil	Incluido	Fossil GUI	En desarrollo (Eclipse Team Provider for Fossil)
Git	gitweb, wit, cgit, GitHub, gitorious, Trac, codeBeamer	gitk, git-gui, tig, TortoiseGit, qgit, gitg, gct, git-cola, Git Extensions, Tower, SourceTree, GitX	Aptana 3 Beta, Eclipse (JGit/EGit), Netbeans (NbGit), Visual Studio (Git Extensions), Emacs, TextMate (Git

			TextMate Bundle), Vim, IntelliJ IDEA >8.1, Komodo IDE; Anjuta
GNU Arch	ArchZoom	ArchWay (GTK2), TlaLog	Emacs
Mercurial	included[107], Bitbucket, Trac, codeBeamer	Hgk, gct, TortoiseHg, MacHg , Murky, SourceTree	IntelliJ IDEA, Eclipse (Mercurial Eclipse), NetBeans, Visual Studio 2008, Emacs, Vim, Komodo IDE
Perforce	Incluido P4Web, P4FTP	Windows, Linux, Unix, Mac OS X, BeOS	Eclipse, Visual Studio, KDevelop, IntelliJ IDEA, Komodo IDE, BBEdit, Emacs
Plastic SCM	SI	plastic	Visual Studio, Eclipse, IntelliJ
IBM Rational Team Concert	SI	GUI basada en Eclipse	Eclipse, MS Visual Studio
Subversion (SVN)	Incluido modulo Apache 2, WebSVN, ViewSVN, ViewVC, Trac, SharpForge, sventon, Warehouse, codeBeamer	Java, KDESVN, Mac OS X, Nautilus, Qt, RabbitVCS, TortoiseSVN	Anjuta, BBEdit, Eclipse, Emacs, IntelliJ IDEA, KDevelop, Komodo IDE, MonoDevelop, Netbeans, RabbitVCS, TextMate, Visual Studio (AnkhSVN, VisualSVN)
Visual SourceSafe (VSS)	SSWI, VSS Remoting	Windows, Linux, Mac OS, Solaris, SourceAnyWhere for VSS	Visual Studio, IntelliJ IDEA

Una de las características más importantes de un sistema de gestión de versiones es la cantidad de interfaces que aporta (Web, GUI, plugins) y que facilitan a los usuarios el proceso de aprendizaje.

En la tabla anterior se puede comprobar como todos los fabricantes facilitan multitud de interfaces para la utilización de su herramienta, tanto para diferentes modos de acceso—Web o de escritorio—como sistemas operativos—*Windows, Linux, Mac OS o Solaris*—.

Pero sin ninguna duda, la característica que más reclaman los desarrolladores de software es la posibilidad de integrar la herramienta de control de versiones a utilizar con el entorno de desarrollo (IDE) al que están acostumbrados. Por ello se puede observar que más del 70% de las herramientas sometidas a estudio incluyen plugins para la integración con *Eclipse* o Visual Studio (IDEs más utilizados del mercado (Hammond, 2008)) y herramientas tan recientes como *Fossil* ya está trabajando en ello.

5.6. Tendencia General de Mercado

A lo largo de este artículo se han presentado diferentes razones que evidencian que un modelo distribuido hace el desarrollo de software más productivo, principalmente por la posibilidad de trabajar a nivel local sin necesidad de conexión con un repositorio central, y esto no solo facilita el trabajo de los desarrolladores sino que también ayuda a la colaboración entre ellos. Además los sistemas distribuidos constituyen un sistema más robusto y menos corrompible, ya que cada desarrollador tiene una copia de toda la historia del repositorio lo que evita que se forme el caos si el servidor central se bloquea o sus archivos se corrompen.

Por todas estas razones y las anteriormente comentadas en este trabajo la tendencia hoy en día es la migración a sistemas de control de versiones distribuidos dejando atrás los viejos sistemas centralizados (Ruparelia, 2010).

Esta tendencia es claramente visible en diferentes estudios y encuestas realizadas sobre el uso de los diferentes sistemas de control de versiones existentes en el mercado. Los datos obtenidos por *Eclipse* en sus informes referentes al año 2009 (Eclipse, 2009) y 2010 (Eclipse, 2010) muestran que el sistema mayoritariamente utilizado sigue siendo *Subversion* (con un 58,3% en 2010 frente al 57,3 obtenido en 2009) y *CVS* se mantiene en segundo lugar, con tendencia decreciente en contraposición a *Subversion*, descendiendo desde el 20% obtenido en 2009 al 12,6% obtenido en 2010. Por otra parte cabe destacar la importancia que están adquiriendo sistemas distribuidos de control de versiones como *Git* o *Mercurial* que han incrementado su popularidad del 2,4% al 6,8% y del 1,1% al 3% respectivamente.

Por otra parte los estudios llevados a cabo por Forrester Research y Dr. Dobbs Developer en sus encuestas Forrester Wave: Standalone SCM Solutions, Q2 2007 (Schwaber, Gilpin, & Stone, 2007), Forrester & Dr. Dobbs Global Developer Technographics, Q3 2009 (Powers, Hill, & Clair, 2009) y Q4 2008 European Application Life-Cycle Management Usage And Trends Online Survey (Hammond, Gilpin, & Sileikis, 2008) demuestran que cuando se trata de sistemas de control de versiones la variedad de herramientas utilizadas de manera habitual por los diferentes desarrolladores es considerablemente elevada, entre herramientas de generaciones antiguas, herramientas de transición y nuevas herramientas que ofrece el mercado. En concreto las encuestas realizadas por Forrester Research encuentran más de 14 sistemas de control de versiones utilizadas hoy en día por los encuestados y son particularmente las herramientas estudiadas a lo largo de este trabajo.

Cabe destacar la conclusión que se puede obtener de los estudios anteriormente mencionados con respecto al aumento de presencia en el mercado actual de *Microsoft*, que ofrece *Microsoft Team Foundation Server* (*Microsoft TFS*) y *Microsoft Visual SourceSafe* (*Microsoft VSS*) con un 49% de los encuestados usando activamente *TFS*, *VSS* o ambos; e *IBM*, que ofrece *Rational ClearCase* y *Rational Telelogic Synergy Suite* con su amplia cartera de herramientas SCCM y consigue un 37% de utilización activa de sus herramientas entre los encuestados.

Finalmente resaltar que ambos estudios dejan de manifiesto que las herramientas de código abierto se han convertido en una atractiva alternativa a las herramientas comerciales y más de la mitad de los encuestados tienen pensado utilizar estas herramientas mientras que sean compatibles con las aplicaciones de gestión del ciclo de vida (ALM) que utilizan. Esta tendencia a favor de soluciones de código abierto es mayor para desarrolladores de Java (un 67%) que para desarrolladores de .NET (un 43%). Es importante tener en cuenta que al referirse a herramientas de código abierto, *Subversion* no es la única alternativa, sino que sistemas como *Mercurial* y *Git* están obteniendo un interés creciente en los desarrolladores.

6. Metodología de selección

Como complemento al análisis y evaluación de las herramientas presentadas en este trabajo, se presenta en este apartado una metodología sencilla a la vez que genérica para la selección de una herramienta de control de versiones por parte de cualquier organización.

Las fases de la metodología son las siguientes:

1. Identificación de las necesidades y caracterización de la organización
2. Búsqueda de soluciones disponibles en el mercado
3. Análisis de características de las soluciones

4. Obtención de soluciones candidatas acorde a las necesidades identificadas en la fase 1
5. Selección final

A continuación se explican brevemente las tareas a desempeñar dentro de cada fase de la metodología:

- Fase 1: Caracterización de la organización y definición de las necesidades

  Es necesario caracterizar a la organización según criterios como el tamaño de la misma, dispersión geográfica y multinacionalidad, número de proyectos desarrollados en paralelo, tamaño medio de los equipos de desarrollo, de los propios proyectos y herramientas de desarrollo utilizadas. Estos parámetros harán necesario o no el soporte multilenguaje y definirán lo importante que puede ser el que la herramienta a adquirir realice las fusiones de los cambios de forma completa o en función de las modificaciones realizadas o la existencia de plugins para las herramientas de desarrollo utilizadas en la organización.

- Fase 2: Búsqueda de soluciones disponibles en el mercado

  Esta fase contempla la búsqueda de herramientas de control de versiones en el mercado. Para ello, como fuentes de información pueden utilizarse Internet (Buscadores, foros, etc.), revistas profesionales, proveedores habituales de la organización y elaborar un listado con todas las soluciones encontradas.

- Fase 3: Análisis de las características de las soluciones

  En esta fase debe realizarse un análisis de las soluciones contenidas en la lista elaborada en la fase anterior. Para ello se puede contactar directamente con los proveedores o consultar foros profesionales donde la información se considere fiable. Los criterios mínimos a comparar son: precio y tipo de licencia, fecha de la última versión, estado del soporte ofrecido por el fabricante, tipo de repositorio (centralizado o distribuido), modelo de concurrencia (LMU o CMM), modelo del almacenamiento de los cambios (changeset o snapshot), alcance del cambio (árbol o ficheros), posibilidad de realización de trazas sobre los cambios, posibilidad para definir diferentes niveles de permisos para los usuarios, soporte para diferentes lenguajes, intrefaz web o gráfico e IDEs para los que ofrece plugin.

- Fase 4: Obtención de soluciones candidatas acorde a las necesidades identificadas en la fase 1

  En esta fase deben enfrentarse las necesidades identificadas en la Fase 1 con las características analizadas en la Fase 3, descartando las herramientas que no representen una solución para la organización.

- Fase 5: Selección final

  Al completar la fase anterior se obtiene como salida un número más reducido de posibles soluciones candidatas. En esta fase, la organización, ponderando en cada caso particular entre las características consideradas más importantes y el precio de estas soluciones, tomará la decisión final sobre la herramienta a utilizar.

7. Conclusiones

La gestión de versiones es uno de los mayores problemas a afrontar en los proyectos software. Para solventar este problema los desarrolladores recurren a las herramientas de control de versiones que facilitan el almacenamiento de los elementos a gestionar, la

recuperación de cada uno de ellos y el registro histórico e identificación de cada una de las modificaciones realizadas en las sucesivas versiones del código del proyecto.

El estado del arte presentado ha puesto de manifiesto que los desarrolladores de proyectos software han preferido herramientas sencillas y más fáciles de usar que los caros, completos y complejos sistemas comerciales sacados al mercado por empresas como *Microsoft* o *IBM*. La reticencia al cambio es otra característica de estos desarrolladores que prefieren seguir utilizando sus arcaicas herramientas con pobre rendimiento en vez de las más novedosas, probablemente por evitar el periodo de transición entre ellas. Esto hace pensar que la creación de una nueva herramienta que realmente sea usada por la gran comunidad de desarrolladores de proyectos software es una labor difícil.

Se ha presentado un estudio de las principales herramientas para realizar la gestión de versiones en proyectos software, analizando tanto sus características como sus funcionalidades y carencias con el fin de comprobar cuales serían las funcionalidades esenciales para la creación de una herramienta idónea que satisfaga las necesidades reales de los equipos de proyecto.

En relación con la facilidad de uso, los desarrolladores valoran positivamente la disponibilidad de diferentes interfaces, no solo a nivel web y de escritorio, sino plugins para la integración de la misma con los entornos de desarrollo habituales. Las nuevas herramientas facilitan desde el primer instante plugins para *Eclipse* y/o *Visual Studio* que hacen su interfaz amigable y lo que es más importante, conocida para el desarrollador.

La tendencia del mercado muestra un giro hacía herramientas open source con licencia GPL que ofrecen las mismas posibilidades que una herramienta comercial por la que se debe pagar un precio alto por usuario. Este giro no solo es provocado por el precio de las herramientas comerciales sino por la cercanía de los desarrolladores de sistemas de control de versiones, que ponen a disposición de los usuarios multitud de blogs y foros donde poder expresar sus disconformidades, problemas o dudas y obtener la respuesta directa de los que más saben de las herramientas concretas.

La asiduidad en la liberación de nuevas versiones también es una de las características destacables que debe poseer una herramienta de control de versiones. Es importante que los fallos (bugs) detectados por los usuarios sean corregidos rápidamente y la nueva versión puesta a disposición de forma inmediata.

8. Referencias

Accurev Inc. (2010). AccuRev Concepts Manual.

Babich, W. A. (1986). *Software configuration management: Coordination for team productivity*. Addison-Wesley (Reading, Mass.).

Bazaar Developers. (2011). Bazaar Tutorial.

Berliner, B., others. (1990). CVS II: Parallelizing software development. *Proceedings of the USENIX Winter 1990 Technical Conference*,341. Citeseer.

BitMover. (2010). BitKeeper User Guide. Retrieved from http://www.bitkeeper.com/UG/Introduction.html.

Bjorkholm, T. (1997). Product Review: Perforce Software Configuration Management System. *Linux Journal*, *1997*(44es).

Cederqvist, P., & Pesch, R. (1993). Version management with CVS. *Signum Support AB*.

Clemm, G. M. (1989). Replacing version-control with job-control. *ACM SIGSOFT Software Engineering Notes*, *14*(7), 162-169.

Collins-sussman, B., Fitzpatrick, B. W., & Pilato, C. M. (2008). Control de versiones con Subversion. *ReVision*.

Eclipse. (2009). The open source developer report 2009 eclipse community survey. Retrieved from http://www.eclipse.org/org/press-release/Eclipse_Survey_2009_final.pdf.

Eclipse. (2010). The open source developer report 2010 eclipse community survey. Retrieved from http://www.eclipse.org/org/community_survey/Eclipse_Survey_2010_Report.pdf.

Estublier, J., Leblang, D., Clemm, G., Conradi, R., Tichy, W., Hoek, A. van der, et al. (2002). Impact of the research community on the field of software configuration management. *ACM SIGSOFT Software Engineering Notes*, *27*(5), 31.

Galfullin, B. N., & Novichkov, A. N. (2000). ClearCase-source code managing system. *Microwave Conference, 2000. Microwave and Telecommunication Technology. 2000 10th International Crimean*, 90-93.

Hammond, J. S. (2008). IDE Usage Trends. *Application Development & Program Management Professionals*.

Hammond, J. S., Gilpin, M., & Sileikis, J. (2008). European Software Configuration Management Tool Adoption Trends.

Hinsen, K., Läufer, K., & Thiruvathukal, G. K. (2009). Essential Tools: Version Control Systems. *Computing in Science & Engineering*, *11*(6), 84-91. Copublished by the IEEE CS and the AIP.

Loeliger, J. (2009). *Version control with Git*. O□Reilly Media, Inc.

Moffitt, N. (2004). Revision control with arch: introduction to arch. *Linux Journal*.

O□Sullivan, B. (2009). *Mercurial: The Definitive Guide*. O□Reilly Media, Inc.

Powers, S., Hill, B. W., & Clair, C. L. (2009). The Forrester Wave™: Enterprise Content Management Suites, Q3 2009.

Pressman, R. S., & Ince, D. (1982). *Software engineering: a practitioner`s approach*. *Techniques*. McGraw-Hill New York, NY.

Rochkind, M. (1975). The Source Code Control System. *IEEE Trans. Software Engineering*, *SE-1*, 364-370.

Roundy, D. (2005). *Darcs: distributed version management in haskell*. *Proceedings of the 2005 ACM SIGPLAN workshop on Haskell - Haskell '05*, 1-4. New York, New York, USA: ACM Press.

Ruparelia, N. B. (2010). The history of version control. *ACM SIGSOFT Software Engineering Notes*, *35*. New York: ACM.

Schimpf, J. (2010). Fossil Version Control A Users Guide.

Schwaber, C., Gilpin, M., & Stone, J. (2007). The Forrester Wave™: Software Change And Configuration Management, Q2 2007.

Serban, A. (2007). *Visual SourceSafe 2005 Software Configuration Management in Practice: Best practice management and development of Visual Studio .NET 2005 applications with this easy-to-use SCM tool from Microsoft*. Packt Publishing.

Software, C. (2011). Introduction to Plastic SCM.

Spinellis, D. (2005). Version Control Systems. *IEEE Software*, *22*(5), 108-109. Published by the IEEE Computer Society.

Sun Microsystems. (2001). Sun WorkShop TeamWare User ' s Guide. *Performance Computing*, *2*.

Team Server. (2008). Quick Start Guide. *International Business*, 2-3.

Tichy, W. F. (1982). Design, implementation, and evaluation of a Revision Control System. *Proceedings of the 6th international conference on Software engineering*, 58–67. IEEE Computer Society Press.

Correspondencia (Para más información contacte con):

Área de Proyectos de Ingeniería – Universidad de Oviedo
Phone: +34 985 104 272
Fax: +34 985 104 256
E-mail: secre@api.uniovi.es
URL: http://www.api.uniovi.es