

Uso practico de CVS para control de versiones
Conceptos y practicas recomendadas

Franco M. Catrin L.

Uso practico de CVS para control de versiones: Conceptos y practicas recomendadas
por Franco M. Catrin L.

Copyright © 2003 4to Encuentro Nacional de Linux

Historial de revisiones

Revisión 1.0 25-02-2003 Revisado por: FCL

Primera version

Tabla de contenidos

1. Resumen	1
2. Introduccion	3
Arquitectura de CVS.....	3
Forma de trabajo basica.....	3
Limites de CVS como herramienta	3
3. Uso basico de CVS	5
Repositorio	5
Modulo.....	5
Trabajo basico con CVS : update y commit	6
Agregando y eliminando archivos	7
Obtencion de diferencias entre repositorio y version local.....	8
4. Caracteristicas avanzadas de CVS	9
Uso de TAG's	9
Uso de ramas (branches)	9
Mezcla de ramas (merge).....	10
5. Practicas recomendadas	11
Uso de ramas estables e inestables	11
Uso de ramas experimentales.....	11
Numero de versiones.....	12
6. Conclusiones	13

Capítulo 1. Resumen

El control de cambios es necesario para el desarrollo de cualquier tipo de proyectos de software. La herramienta mas utilizada para esta labor en proyectos Open Source es CVS (Concurrent Version System). Este documento presenta los conceptos basicos de CVS, asi como tambien características avanzadas y practicas recomendadas para su efectiva utilizacion.

Capítulo 2. Introducción

El desarrollo de software no es una tarea exclusivamente individual. En la gran mayoría de los proyectos tanto Open Source como privados es necesario trabajar en equipos en donde son varios los desarrolladores que acceden al mismo código fuente.

Coordinar las modificaciones concurrentes al código de una forma manual es una tarea titánica, sobre todo en proyectos muy complejos o con una gran cantidad de desarrolladores.

Uno de los objetivos de CVS es proveer una herramienta que permita mezclar distintos cambios al código para consolidar un código único final de una forma automatizada, siempre que sea posible.

Otro de las funciones que cumple una herramienta como CVS es mantener un registro de los cambios que se hacen al código fuente. De tal forma que se tenga acceso a todo el historial de cambios del proyecto. Esta característica permite volver a versiones anteriores, encontrar defectos introducidos y mantener un mismo proyecto en una versión "estable" y "en desarrollo" a partir de un solo repositorio común.

Estas capacidades son de vital importancia en proyectos Open Source, en donde los desarrolladores están separados geográficamente y temporalmente. Sin embargo las mismas prácticas son también aplicables a proyectos privados.

Arquitectura de CVS

CVS es una herramienta que funciona en un esquema de cliente y servidor. Existe un cliente o estación de trabajo en donde los desarrolladores hacen modificaciones al código y realizan las pruebas necesarias para satisfacer los requerimientos. En el servidor existe una versión consolidada del proyecto. Cada cierto tiempo los desarrolladores actualizan sus versiones de trabajo desde el servidor y por otra parte envían sus propios cambios hacia el servidor.

Existen servidores para Linux y Unix, así como clientes para una extensa gama de plataformas (Linux, Windows, MacOSX, etc).

Uno de los beneficios de este esquema, es que se puede integrar la funcionalidad de cliente CVS en las herramientas de desarrollo integradas (IDE). En la plataforma Linux es común que se realice esta integración, debido a lo fundamental que se ha convertido CVS en la mayoría de los proyectos que integran esta plataforma, incluyendo los mismos proyectos de desarrollo de estos IDE's. Es así como por ejemplo podemos encontrar KDevelop de KDE, Anjuta y Eclipse en GNOME integrados (y desarrollados) con CVS

Forma de trabajo básica

Cada archivo que compone el proyecto puede ser modificado por varios desarrolladores al mismo tiempo en forma independiente. CVS se encarga de consolidar los distintos cambios en un solo archivo final que incluya todos los cambios introducidos por los distintos desarrolladores.

Existe solo un caso especial en donde más de un desarrollador puede introducir cambios en la misma parte del código. Esto se conoce como conflicto. Es tarea del usuario que produce el conflicto, resolverlo y enviar una versión consolidada al servidor

Cada archivo tiene un número de revisión independiente. Este número registra el número de cambios hechos al archivo y no tiene ninguna relación con algún número de versión del proyecto completo. Más adelante se verá cómo se pueden manejar los archivos almacenados en CVS como un conjunto

Limites de CVS como herramienta

CVS como herramienta no resuelve todos los problemas del desarrollo en equipo. Simplemente es una ayuda para que las cosas mas tediosas puedan ser automatizadas.

Las areas en donde CVS no esta involucrado tienen relacion con el tratamiento del proyecto a un nivel global. Actividades como la planificacion, los releases, etc, quedan fuera de su ambito de trabajo y deben ser abordadas por las personas y otras herramietnas complementarias

Capítulo 3. Uso basico de CVS

CVS es una herramienta bastante flexible, no es necesario utilizar todas las características que posee, pero si al menos se debe tener conocimiento de las operaciones más elementales para comprender las funcionalidades más avanzadas y que son de interés en este trabajo.

Repositorio

En la parte de servidor de CVS se maneja un repositorio. Un repositorio es simplemente un directorio en el servidor que contiene diversos módulos. Por ejemplo, un proyecto podría tener un repositorio, y en cada módulo estarían los subproyectos.

A su vez, cada módulo contiene un conjunto de archivos, y representa al proyecto con el que se quiere trabajar. Por ejemplo en sitios grandes como sourceforge, cada uno de los proyectos tiene su propio repositorio CVS, y cada uno de ellos tiene uno o más módulos de trabajo. Cuando un desarrollador trabaja con CVS lo hace a nivel de módulo.

Para conectarse al servidor se pueden usar distintos tipos de protocolos, sin embargo los más extendidos son "pserver" para acceso anónimo o algún tipo de acceso en donde la seguridad no es importante. El otro es ssh, y se usa en casos más críticos, por ejemplo cuando se requiere acceso para poder realizar cambios al repositorio.

Para que un directorio pueda ser utilizado como repositorio CVS, lo primero que se debe hacer es decirle a CVS que genere los archivos y subdirectorios necesarios para funcione como tal. Podríamos crear un repositorio con los siguientes comandos:

Primero creamos un grupo especial para que pueda hacer cambios en CVS

```
[root@shaman root]# groupadd cvs
```

Luego se crea un directorio para almacenar el repositorio

```
[root@shaman root]# cd /usr/local/  
[root@shaman local]# mkdir cvs-utfsm
```

Se indica a CVS que inicialice el directorio recién creado

```
[root@shaman local]# cvs -d /usr/local/cvs-utfsm init
```

Se actualizan los permisos para que los usuarios del grupo CVS efectivamente tengan control sobre el repositorio

```
[root@shaman local]# chown root.cvs -R cvs-utfsm  
[root@shaman local]# chmod 775 cvs-utfsm
```

Como resultado se obtendrá un directorio CVSROOT al interior de cvs-utfsm. Este directorio contiene los archivos necesarios para que CVS lo trate como un repositorio

Modulo

Un módulo contiene un grupo de archivos que se administra de manera conjunta. Por ejemplo, cada vez que se realiza alguna actualización a nivel de conjunto de archivos, el ámbito será de un módulo.

Cada repositorio puede tener un número indefinido de módulos, Se sugiere siempre tener los archivos relacionados con un módulo de software bajo un único directorio.

De esta forma, habrá una correspondencia unívoca entre el módulo de software y un módulo CVS.

Para comenzar a trabajar en un módulo a partir del repositorio de ejemplo que se ha creado, lo primero que debemos hacer es importarlo. En este ejemplo se usará una conexión SSH hacia el servidor (aunque sea local).

El primer paso es definir una variable de entorno que para indicar que el protocolo a usar es SSH.

```
export CVS_RSH=ssh
```

Luego se define una variable de entorno para indicar el repositorio que se usará. Esto solo es necesario hacerlo cuando aun no estamos trabajando con ningún módulo.

```
export CVSROOT=:ext:fcattrin@localhost:/usr/local/cvs-utfsm
```

La variable de entorno CVSROOT tiene distintas secciones separadas por ":". La primera sección es el protocolo, en este caso ext indica RSH o lo definido por la variable CVS_RSH.

La segunda sección indica el nombre de usuario con que se accederá al repositorio y el host que lo contiene. Finalmente se indica el directorio del repositorio.

Ahora se puede importar el directorio actual de trabajo en el repositorio. A modo de ejemplo se usará un directorio test contenido en el home del usuario.

```
cd ~/test
cvs import test fcattrin start
```

Este comando crea un módulo llamado test en el repositorio (CVSROOT) usando los contenidos del directorio de trabajo test. Para comenzar a trabajar con este directorio se ejecuta:

```
cd ~
mv test test-sincvs
cvs co test
```

Cuando un directorio se importa como módulo a CVS, las modificaciones solo se realizan en el servidor y el directorio queda tal como estaba en un principio, sin ninguna relación con CVS.

Lo que se hizo fue mover el directorio original a otro sitio y luego efectuar una operación Checkout desde nuestro módulo test recién creado en el repositorio indicado por CVSROOT.

El directorio test volverá a ser creado y contendrá un subdirectorio de control llamado CVS en donde se encuentra la información acerca del módulo correspondiente en el repositorio.

A partir de este momento, ya no será necesario usar la variable CVSROOT al trabajar con este módulo de trabajo, ya que la información de conexión al repositorio estará en el archivo CVS/Root, en el directorio de trabajo.

Trabajo básico con CVS : update y commit

El uso básico de CVS se remite a un pequeño número de tareas, orientadas principalmente a mantener sincronizado el servidor con la versión de trabajo (cliente) y viceversa.

A partir de un modulo que ya existe en nuestro disco, obtenido a traves de la operacion de Checkout (co) indicada en la seccion anterior, podremos comenzar a realizar cambios sobre nuestro software. Los cambios seran registrados en el servidor cuando se considere conveniente. De la misma forma los cambios realizados por otras personas en el servidor seran mezclados en nuestra copia local cuando se desee.

La primera operacion basica sobre nuestra copia local es la actualizacion. Esta operacion revisara que archivos han sufrido cambios en el repositorio respecto a nuestra version de trabajo. Aquellos cambios seran aplicados a la version local, mezclandose con nuestros propios cambios. Tambien podemos indicarle a CVS que genere localmente todos los archivos y directorios que hayan sido creados recientemente por otros desarrolladores.

```
cd ~/test
cvs update -dP
```

Como resultado del comando se vera un listado con cada archivo contenido en el directorio de trabajo. Antes de cada nombre de archivo aparecera un caracter indicando que operacion se realizo:

- ? : Se desconoce el archivo en el repositorio (no hace nada).
- A : Ha sido agregado localmente, pero aun no existe en el repositorio.
- D : Ha sido eliminado localmente, pero aun existe en el repositorio.
- M : Ha sido modificado localmente y/o mezclado con el del repositorio.
- U : Archivo agregado en el repositorio se ha creado localmente.
- C : Hay un conflicto entre los cambios locales y los del repositorio.

El unico caso preocupante es un conflicto. Como consecuencia de el, en el codigo quedara una seccion marcada como conflicto en donde apareceran dos versiones. En una estaran los cambios realizados localmente, y en la otra los cambios realizados remotamente. El usuario debe resolver este conflicto editando el archivo para dejar la version final.

Cuando ya se han realizado todos los cambios necesarios, se deben subir al repositorio estos cambios. Esto se realiza con la operacion Commit (ci) que puede operar sobre un grupo de archivos o sobre el modulo completo.

Al realizar un commit es obligatorio dejar un mensaje explicativo para que quede registrado junto al cambio en CVS. Esto es muy util para obtener un historial de cambios al proyecto. Para poner el mensaje se debe usar la opcion -m

Ejemplos de commit:

```
cvs ci -m "Solo cambios a un solo archivo" archivo.c
cvs ci -m "Cambios globales"
```

El primer ejemplo solo actualiza el "archivo.c", el segundo ejemplo actualiza el modulo completo.

Puede darse el caso de que existan revisiones mas nuevas de los archivos modificados. Ese caso aparecera un error de "up-to-date failed" en el cliente y debera realizarse un update en el repositorio local, antes de poder efectuar el commit.

En el ciclo de vida de un proyecto de software, la mayor parte del tiempo solamente sera necesario usar comandos de update y commit.

Agregando y eliminando archivos

Otras operaciones básicas son la agregación y eliminación de archivos al repositorio. En un principio, CVS maneja todos los archivos existentes al momento de realizar la operación de import del directorio en el repositorio. A partir de ese momento, cualquier cambio en los archivos manejados por CVS, por ejemplo agregar y/o eliminar, deberá realizarse de forma explícita.

Para agregar un archivo se debe utilizar la operación "add" sobre el archivo o directorio. Los archivos quedarán marcados para agregación, pero esta operación solo se hará efectiva al momento de hacer un commit.

Para eliminar un archivo, primero debe eliminarse físicamente del directorio, luego se usa la operación "remove" sobre el nombre del archivo y este quedará marcado para su eliminación en CVS. Al igual que "add", la operación se hará efectiva al hacer commit.

En el siguiente ejemplo se agregará el archivo nuevo.c y se eliminará el archivo antiguo.c:

```
cvs add nuevo.c
rm antiguo.c
cvs remove antiguo.c
cvs commit
```

Obtención de diferencias entre repositorio y versión local

A veces es necesario conocer qué cambios se han realizado a nivel local o en el repositorio. Por ejemplo para auditar nuestros propios cambios o para revisar los cambios que se integrarían en caso de efectuar un update.

Otro uso interesante, sobre todo cuando se trabaja con proyectos ajenos es obtener un archivo de "patch" para enviar a los desarrolladores del proyecto con acceso de escritura al CVS. Esto es fundamental en el mundo del Open Source para poder filtrar los distintos aportes que se hacen a un proyecto desde gente externa a él.

El comando a usar es "diff", y se puede realizar sobre un archivo o sobre el repositorio completo. Incluso se puede realizar entre nuestro archivo local y alguna revisión específica de ese archivo en el repositorio.

Ejemplos

```
cvs diff nuevo.c
cvs diff -r1.1 nuevo.c
cvs diff
```

Lo que hacen estos comandos es mostrar las diferencias de la versión de trabajo respecto a la versión del repositorio del archivo nuevo.c. El segundo comando compara el archivo local respecto a la revisión 1.1 de este archivo en el repositorio. Y el último comando busca todas las diferencias entre el directorio de trabajo y el módulo en el repositorio.

Capítulo 4. Características avanzadas de CVS

Como se dijo en un principio, CVS puede usarse para administrar los cambios hechos a un proyecto, pero también puede utilizarse para poder cosas más interesantes como por ejemplo realizar distintos desarrollos en paralelo.

Uso de TAG's

CVS mantiene un registro de los cambios realizados por archivo. Cada archivo tiene un número de revisión que representa el número de cambios realizados.

No existe una forma en que CVS maneje automáticamente algún número de versión para el módulo completo en un estado determinado. Menos aún alguna forma de relacionar esta versión con las de otros módulos.

Por lo tanto, en cuanto a número de versión, CVS verá un conjunto de archivos, cada uno con su propio número de revisión sin ningún orden en particular.

Para poder marcar un estado del repositorio se deben usar TAG's. Un TAG es un nombre que se asigna a todos los archivos del módulo en un instante determinado. Este TAG permite obtener versiones del módulo independiente de los números de revisión de los archivos.

Usando TAG's se puede marcar por ejemplo el estado del repositorio en el momento de hacer un release del módulo. Los nombres usados por los TAG's son arbitrarios, así que también se le puede dar cualquier otro uso que se estime conveniente.

Un ejemplo de marca de TAG sobre el módulo sería:

```
cvs tag nombre-de-tag
```

Un TAG realmente es un nombre que se le pone a un número de revisión de un archivo, lo que hace el comando es poner el TAG a todos los archivos en su número de revisión actual.

También es posible mover un TAG, es decir, aplicar el nombre sobre un conjunto de archivos con sus números de revisión más recientes. Ejemplo

```
cvs tag -F nombre-de-tag
```

La ventaja de mover un TAG se puede ver a la hora de marcar un release. Por ejemplo si algún cambio se quedó fuera del release, puede aplicarse y luego mover el TAG al estado actual.

Uso de ramas (branches)

Uno de los problemas de trabajar con versiones residentes en CVS, o en cualquier otra forma en estado "cambiante", es que no siempre se puede asegurar que el módulo se encuentre en un estado usable o estable. En cierta forma el uso de TAG's permite acceder a ciertos estados transitorios, pero cuando se quieren hacer cambios profundos que necesitan varios cambios en el repositorio, ya no es tan efectivo.

Para este tipo de caso existen los branches. Un branch se puede ver como una línea de evolución de un módulo. Por defecto se trabaja en un solo branch llamado HEAD. En ese branch hay una evolución del módulo, pero solo existe un antes y un ahora, pero no pueden haber desarrollos aislados o paralelos.

Al usar branches adicionales se puede generar una línea paralela de desarrollo del repositorio CVS. Esta línea paralela se inicia en un estado determinado del repositorio y luego sigue su camino independiente con su propio conjunto de cambios y

numeros de revisiones. Se pueden hacer todos los cambios que se deseen en la línea paralela, así como en la línea de desarrollo original, sin que interfieran entre ellos.

Esta característica puede permitir que un equipo se encuentre trabajando en características nuevas, mientras que otro equipo trabaja en bug-fixes de una versión estable.

Cada rama tiene un nombre definido por el usuario, salvo la rama principal que se conoce como HEAD. Un ejemplo de creación de una rama es:

```
cvs tag -b nombre-de-rama
```

Después de crear la rama, el directorio local aún seguirá en su rama original. Para cambiarse a la rama, se debe usar un update especial. Ejemplo:

```
cvs update -r nombre-de-rama
```

Una vez que se está en una rama, todos los commit's se harán en esa rama y las otras no podrán ver estos cambios. Se puede cambiar el directorio local a cualquier rama que se desee, preservando los cambios hechos a la copia local.

Mezcla de ramas (merge)

Como se ha dicho anteriormente, todas las ramas tienen su entorno propio sin afectar al resto. Pero frecuentemente llega un momento en que los cambios realizados en una rama, deben aplicarse sobre otra rama para pasar a formar parte de una línea común de desarrollo.

A esto se le llama merge o mezcla de ramas. Su forma de operar es muy similar a realizar un update desde el repositorio. Es decir, se producen actualizaciones, mezclas y en algunos casos, conflictos.

Para mezclar una rama con otra, primero se debe estar en la rama de destino, o la rama que consolidará los cambios. Luego se utiliza un comando para obtener los cambios desde otra rama. Ejemplo:

```
cvs update -j nombre-de-rama
```

Capítulo 5. Practicas recomendadas

En el uso normal de CVS, el uso basico explicado al inicio de este documento no hay mucho que agregar. El procedimiento es simple y se sigue un ciclo de desarrollo normal en comparacion a un desarrollo sin CVS. Las ventajas se distinguen mas cuando se comienzan a usar branches.

Se puede definir una metodologia para mantener un estado sano de los repositorios CVS, de tal forma que siempre se pueda obtener una version estable del CVS, y ademas dar flexibilidad para que se puedan estar haciendo cambios en el CVS sin alterar las versiones estables.

Que se gana con esta separacion de estados del repositorio. Supongamos que se ha trabajado duro en un proyecto y ya se ha hecho un release estable. La comunidad o incluso un cliente podria comenzar el uso de esta version en un entorno real.

Sin embargo se quiere seguir desarrollando el proyecto, y se necesitan cambios profundos al codigo actual, asi como tambien se necesitan hacer bugfixes a la version liberada como estable. En una forma de trabajo lineal con CVS esto no seria posible, ya que los bugfixes se aplicarian sobre la version actual del proyecto, es decir, la version en desarrollo. En este caso no se podria entregar una version al exterior que no incluya todos estos cambios profundos.

Justamente el uso de ramas permite solucionar este problema

Uso de ramas estables e inestables

A lo largo de la vida de un proyecto, se pueden tener distintos releases de versiones estables, y releases de versiones inestables o en desarrollo. Asi como bugfixes en el caso de la version estable y construccion en la version inestable.

Una practica comun es usar la rama principal o HEAD como linea de desarrollo. En esta rama encontraremos la ultima version del software y estara sujeta a todos los cambios requeridos en su construccion. Los desarrolladores trabajan principalmente sobre esta rama.

Cada vez que se genere un release inestable. Se puede usar un TAG para marcar el estado del repositorio en el momento del release. De esta forma, se puede obtener desde CVS exactamente la version liberada en un momento determinado.

Al momento de hacer un release estable. Tambien se puede aplicar un TAG para marcar el estado, pero ojo, posteriormente se necesitan hacer bugfixes a esta version del repositorio, independiente de lo que haya en desarrollo. Entonces, es el momento de crear un branch estable de CVS.

Al crear un branch, el repositorio se separara en una linea de bugfixes en la rama recién creada y la linea normal de desarrollo en la rama principal. De esta forma, siempre es posible obtener una ultima version del release estable con todos sus bugfixes utilizando la nueva rama, y en forma separada la ultima version en desarrollo por medio de la rama principal.

Uso de ramas experimentales

A veces se necesitan hacer cambios profundos al codigo durante periodo prolongado de tiempo, esto haria que la version de CVS podria estar totalmente rota, independiente de estar trabajando en una rama estable o inestable.

Este tipo de cambios puede ser parte de un nuevo desarrollo o por ejemplo una optimizacion profunda que debe ser muy bien probada antes de ser parte del codigo oficial. En casos como este tambien se puede aplicar una rama.

Esta rama no tiene ninguna diferencia con las creadas anteriormente, salvo el uso que se le va a dar. Al crear una rama, se pueden hacer todos los cambios que sean

necesarios sin afectar al resto del equipo que se encontrara trabajando en alguna otra rama. Solamente cuando esten los cambios lo suficientemente probados, se podria hacer la mezcla en su rama original.

Tambien podria darse el caso de que el trabajo no haya dado su fruto esperado, y por lo tanto nunca deba formar parte del codigo oficial. La rama puede quedar abandonada sin problemas, y quedara simplemente registrada en la historia del proyecto.

Numero de versiones

Los numeros de versiones usados en los releases no tienen relacion directa con CVS, ya que es algo que pertenece al dominio del proyecto y no al dominio de CVS como herramienta. Sin embargo, se pueden usar algunas convenciones para saber siempre con que parte del proyecto estamos trabajando.

La mayoría de los proyectos Open Source tienen numeros de versiones de tres digitos, organizados en un formato A.B.C. Para explicar el significado de cada digito, se pueden explicar de derecha (C) a izquierda (A) de la siguiente forma:

- C : se incrementa cuando se hacen bugfixes y/o cambios pequeños. Agrupa varios cambios en un solo cambio lógico a nivel de proyecto
- B : cambia cuando se hacen cambios significativos y que significan una real mejora a la versión anterior. Este valor puede ser de dos tipos:
 - PAR: indica que la version es considerada estable hasta donde se sabe. Se mantiene en su propia rama de release estable
 - IMPAR: indica que la version no es considerada estable, ya que se encuentra en desarrollo. Estas versiones estan pensadas para liberar ultimos desarrollos que pueden ser utiles al resto del equipo, pero que aun no han madurado lo suficiente para llevarlos a producción. Se mantienen en la rama principal
- A : indica cambios radicales en el software, como son los cambios de diseño que lo hacen en cierta forma incompatible con la version anterior

Usando este esquema de numeracion, un proyecto partiria en la rama principal con un numero de version 0.1.0. A medida que se hacen releases de esta version de desarrollo, la numeracion ira avanzando a 0.1.1, 0.1.2, etc. En general 0.1.x.

Llegara un momento en donde es posible hacer un release estable, y tendremos una version del repositorio CVS con dos numeros de release, uno estable y otro inestable. Esto se debe a que el release estable corresponde al mismo codigo de uno inestable. Por ejemplo, si tenemos la version 0.1.8 y esta ya puede ser considerada estable, tendremos el release estable 0.2.0 con el mismo codigo de la 0.1.8.

A partir de ese release se creara una rama para seguir la historia de 0.2.x, y esta rama tendra las versiones 0.2.1, 0.2.2, etc.

Por otra parte la rama de desarrollo pasara de 0.1.8 a 0.3.0. Seguira evolucionando la rama 0.3.x hasta el proximo release estable 0.4.0.

Capítulo 6. Conclusiones

El rápido avance de los proyectos Open Source no solo se debe al gran entusiasmo y esfuerzo de los distintos desarrolladores alrededor de todo el mundo. Parte del logro también se debe al uso de herramientas de desarrollo como es el caso de CVS entre otras.

El uso de CVS permite automatizar la compleja tarea de administración de cambios sobre todo el proyecto, incluyendo la evolución en líneas paralela de un mismo proyecto.

Esta forma de trabajo no solo es aplicable a proyectos Open Source, sino que también puede mejorar bastante los procesos de desarrollo de software en empresas que generan software cerrado, permitiéndoles siempre tener el control sobre sus proyectos en desarrollo y la mantención de releases declarados como estables.

