CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UD 3.1

CONTROL Y GESTIÓN DE VERSIONES

ENTORNOS DE DESARROLLO 20/21
CFGS DAW

PARTE 1 DE 4: HERRAMIENTAS CASE Y CONTROL DE VERSIONES

Autora: Cristina Álvarez Villanueva
Revisado por: Sergio Badal
sergio.badal@ceedcv.es
Fecha: 28/10/20

ÍNDICE DE CONTENIDOS

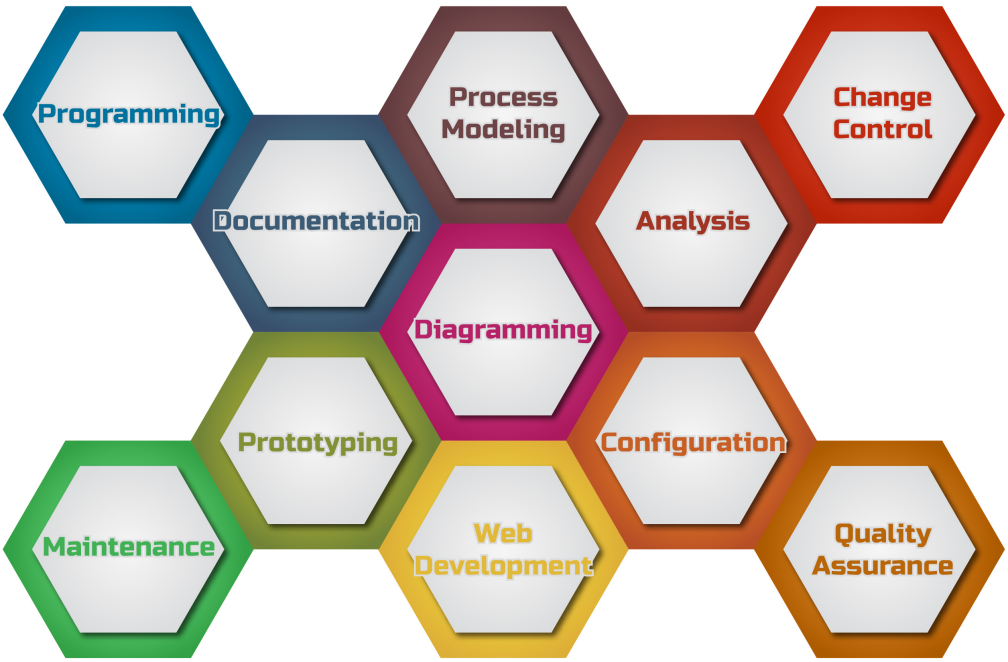
1. HERRAMIENTAS CASE

Hace años un ~~programador~~ desarrollador era una persona que, habitualmente en solitario, desarrollaba aplicaciones monolíticas con un número reducido de ficheros y una vida útil corta. Hoy en día, dado el expansivo uso de los ordenadores y gracias a la ubicuidad que ofrece Internet, el desarrollo de aplicaciones ha tenido un crecimiento exponencial y la programación se realiza de manera colaborativa, incluso entre equipos de programadores de todo el mundo.

La vida del "objeto programado" es ahora larga ya que se desea amortizar la inversión realizada. Por este motivo, los ~~programadores~~ desarrolladores fueron diseñando **herramientas que les ayudaran en cada una de las fases del ciclo de vida de una aplicación**.

De este modo, existen CASE para cada una de las fases del ciclo de vida del software pudiendo aplicarse algunas de ellas en una o más fases. En concreto, **la fase de codificación es la que tiene más herramientas** asociadas y lo habitual es tenerlas integradas en un IDE (Integrated Development Environment). Estos entornos de desarrollo son aplicaciones modulares que, con un editor de código como elemento central, incluyen pluggins adicionales.

Este tipo de herramientas se conocen como **herramientas CASE (Computer-Aided Software Engineering)** y son aquellos instrumentos software que apoyan al desarrollo de una aplicación.



Algunos de los CASE más empleados son:

1. entornos de desarrollo (como Visual Studio Code)
2. compiladores (como javac)
3. depuradores (como Eclipse)
4. refactorizadores (como NetBeans)
5. analizadores de rendimiento (como JetProfiler)
6. herramientas de análisis y diseño (como DIA)
7. herramientas de despliegue e instalación (como Bamboo)
8. editores de código (como Notepad++)
9. generadores de código (como el de Ruby On Rails)
10. generadores de documentación (como javadoc)
11. gestores de proyectos (como Microsoft Projects)
12. gestores de incidencias (como Mantis)
13. control de versiones (como GIT)

RETO

Los más nuevos en esto de "la programación" solo piensan en los IDE cuando hablamos de herramientas CASE. ¿Conoces alguna herramienta CASE que no pueda agruparse en ninguna de las categorías anteriores? ¿Has usado alguna en concreto, esté o no en estas categorías?

Cuéntanos tu experiencia con "las CASE" en el FORO DE LA UNIDAD…

2. SISTEMAS DE CONTROL DE VERSIONES (VCS)

2.1 QUÉ ES UN VCS

Un sistema de control de versiones, **VCS (Version Control System)** es una herramienta CASE que se emplea en la fase de integración y/o mantenimiento y **puede venir o no incluida en los IDE**.

Durante el desarrollo de un proyecto, muchos de los archivos asociados a él irán evolucionando: se añadirán y modificarán.

- En el caso de un programador solitario, en su día a día hace cambios de su código frecuentemente (corrige código, añade nuevas características o lo modifica). Cada vez que se da un nuevo cambio, la antigua versión del fichero se sobrescribe. Pero si este código genera error, **es interesante poder volver a la versión anterior para comprender qué ha ocurrido**.
- Si esto se amplía a un escenario de trabajo colaborativo, donde dos programadores están trabajando sobre un mismo fichero compartido en red, la situación se complica.

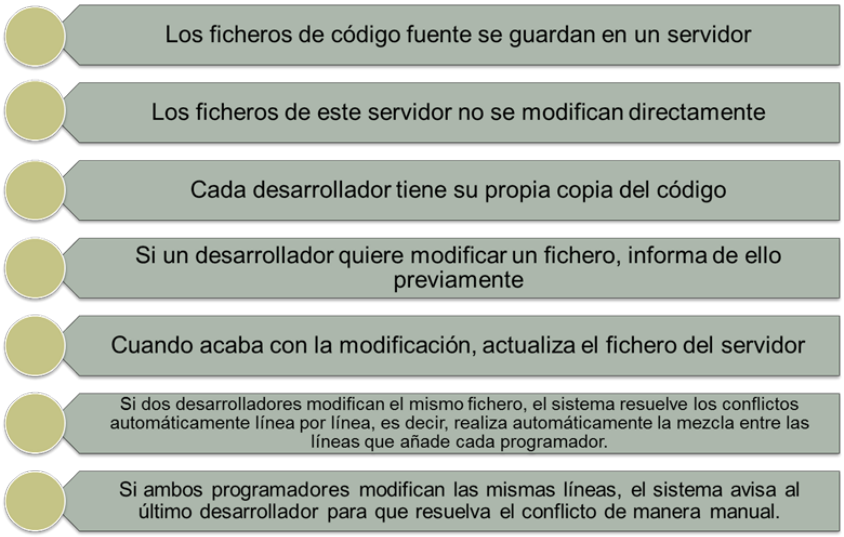
> Interesante
>
> **Este concepto no es exclusivo del desarrollo de software, todos estamos acostumbrados a diferentes ediciones de libros, revisiones de coches, televisores, etc.; sin embargo ha tomado mucha más relevancia en la era digital. Programas como Microsoft Word nos permiten gestionar diferentes versiones del mismo documento, compararlas, etc. Incluso los sistemas operativos van introduciendo poco a poco funcionalidades en esta dirección.**

En definitiva, en el desarrollo de software es vital poder controlar diferentes versiones de los mismos documentos. Los sistemas de control de versiones realizan estas tareas y además ofrecen otras funcionalidades, como comparar versiones, etiquetar, crear bloqueos, etc.

Generalmente son aplicaciones específicas que permiten especificar qué y cómo controlar. Existen muchas opciones en este tipo de software pero los más extendidos en la actualidad son Subversion (Apache Subversion o SVN) y GIT.

2.2 FUNCIONAMIENTO GENERAL

Aunque veremos que el funcionamiento de un VCS depende del tipo que sea, en general tienen las premisas que se indican a continuación:



2.3. USOS DE UN VCS

Un VCS debería usarse siempre. Las situaciones en que podríamos haber necesitado usarlo son:

- Si tras hacer un cambio en el código surge un error y queremos volver a la versión anterior
- Si queremos recuperar código perdido y nuestra copia es demasiado antigua
- Si queremos estar al día de la última versión del desarrollo si trabajamos en equipo
- Si queremos mantener varias versiones del mismo producto
- Si tenemos dos versiones del código y queremos ver las diferencias
- Si queremos realizar pruebas de errores en la aplicación
- Si queremos probar código que ha hecho otro programador
- Si queremos se hizo un cambio y queremos saber qué, cuándo y dónde
- Si queremos experimentar con nuevas características sin interferir en el código

Siendo imprescindible en entornos colaborativos, **prácticamente no se usa en individuales**.

> Importante
>
> **Cuando se usa en entornos individuales suele usarse como una mera copia de seguridad de nuestros ficheros cuando es justamente a la inversa como debemos usarlo, debiendo incluir los datos de nuestro VCS en nuestra copia de seguridad.**

2.3 VOCABULARIO BÁSICO

Los conceptos fundamentales de todo VCS son:

Repositorio *(repository)*	• base de datos donde se guardan los ficheros
Servidor *(server)*	• ordenador donde se guarda el repositorio
Cliente *(client)*	• ordenadores que están conectados al repositorio
Copia de trabajo o de directorio *(working copy or workings setk)*	• directorio local (en clientes) en el que se hacen los cambios
Tronco o principal *(trunk or main)*	• línea principal de código en el repositorio
Cabecera *(head)*	• última versión en el repositorio
Revisión *(revision)*	• versión en la que se encuentra un fichero
Rama *(branch)*	• copia separada de un fichero o una carpeta para uso privado
Conflicto *(conflict)*	• situación que ocurre cuando se intentan aplicar dos cambios contradictorios de un mismo fichero sobre el repositorio
Mensaje de registro *(checkin message)*	• mensaje corto que indica qué se ha cambiado en el fichero
Histórico de cambios *(changelog or history)*	• listado de todos los cambios realizados en el fichero desde su creación

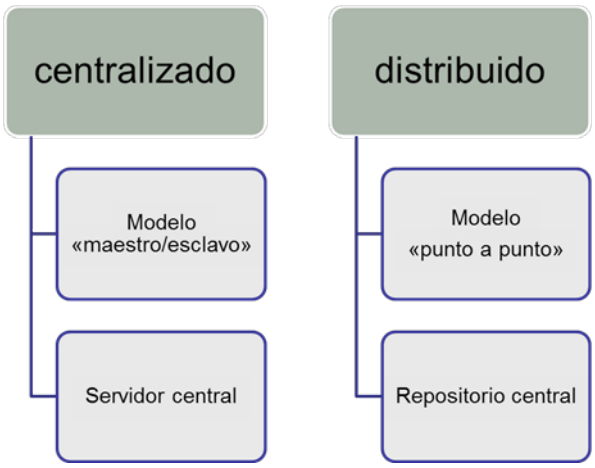
En cuanto a las acciones, las más habituales son:

add	• Añadir un fichero al repositorio por primera vez para permitir al sistema hacer su seguimiento
get latest/check out	• Traer desde el repositorio la última versión del fichero
check out for edit	• Traer desde el repositorio la última versión del fichero en modo "editable". En muchos sistemas el check out ya es editable.
check in/commit	• Subir un fichero modificado al repositorio. Se le asigna un número de versión y los demás programadores pueden hacer un check out o un sync para obtener la última versión
merge	• Mezclar los cambios de un fichero a otro para actualizarlo.
resolve	• Resolver los problemas encontrados al hacer check in. Una vez solucionados, se hace el check in de nuevo.
diff /delta	• Comparar dos ficheros para encontrar las diferencias
revert	• Revertir la última versión: se descartan los cambios locales y se recarga la última versión que existe en el repositorio
update/syc	• Actualizar todos los ficheros con la última versión del repositorio
locking	• Bloquear un fichero, tomando su control para que nadie más pueda editarlo hasta desbloquearlo
breaking the lock	• Forzar el desbloqueo de un fichero

2.4 TIPOS DE CONTROL DE VERSIONES

Hay dos tipos de VCS:

- Centralizados
- Distribuidos



En ambos, el usuario tiene un directorio de trabajo en su disco duro local en el cual tiene una copia de código y un repositorio con la historia de todo el proyecto registrada. Cuando se hace un cambio en la copia local se debe enviar al repositorio. Veamos en detalle cómo funciona esto para cada tipo de VCS.

2.4.1 VCS CENTRALIZADO

Los sistemas de control de versiones centralizados emplean un **servidor central** para guardar el repositorio y el control de acceso al mismo. Este repositorio está separado de la copia que existe en el directorio de trabajo de cada usuario. El servidor debería ser **dedicado** aunque a nivel individual también puede ser una carpeta del mismo ordenador de trabajo. Sin embargo, por seguridad, es conveniente que el servidor sea instalado en un ordenador del equipo de desarrollo distinto a los empleados para programar.

La copia de trabajo sólo almacena la versión actual de los ficheros del proyecto y las modificaciones del momento. Cuando un usuario envía cambios (hace un check in), los manda al repositorio central. En cuanto llegan, podrán ser descargados por el resto de usuarios. Esta metodología sigue el sistema **maestro/esclavo**.

2.4.2 VCS DISTRIBUIDO

En este tipo de sistemas **cada usuario tiene su propio repositorio.** Los cambios que haga un usuario en su repositorio local podrán ser compartidos o no con los demás.

Todos los usuarios son iguales entre sí, con lo que no se tiene el concepto maestro/esclavo del centralizado. **Esta metodología de trabajo se conoce como punto a punto.**

Para evitar el caos en las copias, se suele usar un **repositorio central** añadido donde sincronizar todos los cambios locales del usuario. Este repositorio hace las veces de centralizado pero por una decisión logística o política, no por necesidades técnicas como en el caso de un VCS centralizado.

Otra diferencia con el sistema centralizado es que cada uno de los usuarios puede hacer un check in y un check out de su propio repositorio, teniendo acceso a estos cambios solamente este usuario. Puede decidir

posteriormente compartirlos con los demás o con un servidor. (Recordemos que en sistema centralizado todos los usuarios tienen acceso a cualquier modificación)

Nosotros vamos a trabajar con un sistema distribuido.

Veamos sus ventajas e inconvenientes:

Ventajas:

- El ordenador de cada usuario hace de entorno de pruebas.
- Permite trabajar sin conexión. Sólo hay que conectarse cuando se quieran compartir los cambios.
- Es rápido. Los cálculos de las diferencias se hacen en local
- Se asigna un identificador por cada cambio, con lo que es fácil hacer un seguimiento del mismo
- El trabajo con el repositorio local permite la creación de ramas y un mezclado muy sencillo. Además, se pueden crear ramas en local para pruebas personales así como tener un servidor remoto para crear una nueva revisión de la aplicación
- Requieren menos mantenimiento. En general, no es necesario instalar y configurar el servidor
- Tiene mayor seguridad ante pérdidas de información, ya que al estar replicada en repositorios locales permite volcar uno de ellos en caso de problemas.

Desventajas:

- El hecho de que sea más seguro no significa que sirva de copia de seguridad. De hecho, los cambios no replicados se perderían en caso de fallo físico del sistema.
- No queda claro cuál es la última versión estable del código a no ser que haya repositorio central.
- No existen números de revisión definidos: los números que se asignan a cada cambio son en realidad números locales.

Con lo que no es un número de versión real. Sin embargo, sí existe un identificador de la forma e4e561f3 por cada cambio que es difícil de recordar.

En un sistema distribuido tenemos terminología específica que se añade a la ya vista:

push	• (empujar) Envía un cambio desde un repositorio a otro.
pull	• (extraer) Coge los cambios desde un repositorio
clone	• (clonar) Trae una copia exacta del proyecto desde un repositorio a otro

2.5 CONCLUSIONES

El funcionamiento de un VCS depende de su tipo, pero de manera general se cumple que:

1. Se almacenan los ficheros del código fuente en un servidor
2. Los ficheros de ese servidor no se modifican directamente
3. Cada desarrollador tiene su propia copia del código
4. Si un desarrollador quiere modificar un fichero informa que va a realizar dicha modificación
5. Cuando la modificación está realizada, actualiza el fichero del servidor
6. Cuando dos usuarios modifican el mismo fichero, el sistema resuelve los conflictos automáticamente línea por línea, es decir, si un programador modifica las líneas superiores del fichero y otro las inferiores, el sistema automáticamente realizará la mezcla
7. En el caso de que ambos programadores modifiquen las mismas líneas, el sistema avisa para que el último desarrollador resuelva el conflicto de manera manual

2.6 EJEMPLOS

Existen múltiples VCS en el mercado. Algunos de los más conocidos son:

- **Concurrent Versions System (CVS)**: fue el originario. Es de código abierto y modelo centralizado. Es muy estable pero lleva sin sacar una nueva versión desde 2008.
- **Subversion (SVN)**: es la evolución de CVS. Fue junto con Mercurial uno de los estándares de facto del modelo distribuido durante muchos años.
- **Visual SourceSafe y Visual Studio Team Foundation Server**: software de Microsoft. El primero es para equipos de desarrollo pequeños. El segundo es una versión mejorada que permite, además del control de versiones, controlar incidencias en la gestión proyectos en equipos entre otras. Son de código propietario y de modelo centralizado.
- **BitKeeper**: fue usado durante años para el control del Kernel de Linux. Es código propietario y sigue el modelo distribuido.
- **Mercurial**: surgió como alternativa a BitKeeper. Es considerado un estándar de facto en modelos distribuidos.
- **GIT**: alternativa a Mercurial. De código abierto y modelo distribuido (Nosotros vamos a trabajar con GIT)

3. BIBLIOGRAFÍA

i. Aldarias, F. (2012): “Entornos de desarrollo”, CEEDCV

ii. Casado, C. (2012): *Entornos de desarrollo*, RA-MA, Madrid

iii. Oltra, A. (2013): “Sistemas de control de versiones. Git”, CEFIRE, Valencia Ramos, A.; Ramos, MJ (2014): *Entornos de desarrollo*, Garceta, Madrid