# Tema 6: Documentación y control de versiones.

"La programación es una carrera entre ingenieros de software luchando para construir programas cada vez más grandes, mejores y a prueba de idiotas, y el universo intentando producir cada vez más grandes y mejores idiotas. Por ahora, gana el universo. " Rich Cook.

## 1.- Documentación de aplicaciones web.

¿Qué conviene documentar en una aplicación? Tres aspectos fundamentales de la aplicación:

* **La interfaz:** qué hace (no como lo hace) una función o un método de una clase, qué parámetros hay que pasar y qué devuelve.
* **La implementación:** indicar cómo está implementada cada función, cómo se lleva a cabo cada paso, por qué se utiliza determinada variable, qué algoritmo se utiliza, qué hacen los métodos privados de una clase.
* **La toma de decisiones:** por qué se ha implementado de determinada forma y no de otra la aplicación.

Normalmente la información sobre la implementación no necesita salir del código pero, por el contrario, la información de la interfaz conviene pasarla a un documento indenpendiente del código fuente (manual de uso).

La persona que necesite utilizar una determinada librería de clases o funciones tendrá toda la información necesaria: qué hace cada elemento y cómo se utiliza. No necesita acceder al código fuente.

Debido a actualizaciones, correciones, etc se necesita automatizar el proceso.

Herramientas que permiten generar documentación de forma automática a partir del código fuente:

- **Javadoc** herramienta estándar para Java.

- **phpDocumentor** herramienta para PHP.

Los entornos de programación modernos como **NetBeans** o **Eclipse** aprovechan los comentarios de nuestro código fuente para mostrar información muy útil.

Estas herramientas, esperan el mismo tipo de comentarios, basado en el **estándar** establecido por JavaDoc, de modo que haremos el trabajo una sola vez y podremos aprovecharnos del mismo en varios entornos y con varias herramientas.

## 2.- PhpDocumentor.

Herramienta más utilizada para php.

La documentación de un proyecto de software es tan importante como su código.

Una buena documentación nos facilita, en gran medida, el mantenimiento futuro de la aplicación.

Y de gran utilidad si trabajamos en equipo.

phpDocumentor, genera automáticamente documentación de nuestro código, mediante comentarios y unas etiquetas especiales podemos definir de forma sencilla qué hace cada clase, cada método y cada función de nuestro código.

phpDocumentor (software libre) permite generar la documentación de varias formas y en varios formatos.

* Desde línea de comandos (CLI).
* Desde interfaz web.
* Desde código, scripts.
* Integrado con los IDEs de desarrollo en PHP (NetBeans, VSC, Eclipse).

Parámetros específicos:

1. El directorio en el que se encuentra nuestro código.
2. Opcionalmente los paquetes (@pakage) que deseamos documentar, lista de ficheros incluidos y/o excluidos y otras opciones interesantes para personalizar la documentación.
3. El directorio en el que se generará la documentación.
4. Si la documentación va a ser pública (sólo interfaz) o interna (en este caso aparecerán los bloques private y los comentarios @internal).
5. El formato de salida de la documentación.

Formato de salida:

* **HTML** a través de un buen número de plantillas predefinidas.
* **PDF**
* **XML** (DocBook), a partir de este podemos transformar (XSLT) a cualquier otro utilizando nuestras propias reglas y hojas de estilo.

Alternativa **Doxygen** la diferencia principal es que es un programa, mientras que phpDocumentor es una colección de código en PHP es por eso que se necesita tener PHP instalado.

## 2.1.- Instalación de phpDocumentor

* Instalación de Ubuntu 20.04 o superior
* Instalación de Xampp (el interprete de php esta en: **/opt/lampp/bin/php**.)
* Instalar php, php-xml
* Hacer una prueba: <http://localhost>
* CLI: php -r “phpinfo();”
* En /opt/lampp/htdocs : crear phpinfo.php
* Descargar el paquete mediante: wget <https://phpdoc.org/phpDocumentor.phar>
* Luego:
  + Chmod +x phpDocumentor.phar
  + Sudo mv phpDocumentor.phar /usr/local/bin/phpdoc
* Manera global: phpdoc run –d . –t docs/api
* Este comando genera una estructura de documentación vacía en la carpeta docs/api.

## 2.2.-Funcionamiento de phpDocumentor

La documentación se distribuye en bloques “DocBlock”. Estos bloques siempre se colocan justo antes del elemento al que documentan y su formato es:

|  |
| --- |
| /\*\*  \* texto  \* mas texto  \*/  function suma(){  } |

Los elementos que pueden ser documentados son:

* Function
* Costant
* Class
* Interface
* Trait
* Class constant
* Property
* Method

Se puede incluir documentación global a nivel de fichero y clase mediante **@package.**

Dentro de cada **DockBlock** se pueden incluir marcas o etiquetas:

* @author
* @copyright
* @deprecated
* @example: permite especificar la ruta hasta un fichero con código PHP.
* @ignore: evita que phpDocumentor documente un determinado elemento.
* @internal: documentación interna, NO es pública.
* @link: enlace <http://> a un determinado recurso.
* @see:crea enlaces internos.
* @since: indica que el elemento está disponible desde una determinada versión del paquete o distribución.
* @version

Marcas que solamente se pueden utilizar en bloques de determinados elementos:

* @global: para especificar el uso de variables globales dentro de una función.
* @param: para documentar parámetros que recibe una función.
* @return: valor devuelo por una función.
* @throws: indica si el método o la función puede lanzar algún tipo de excepción.

Crear proyecto. Y almacenarlo en **/opt/lampp/htdocs**

* **Instalar xampp**
* **Sudo /opt/lampp/lampp start**
* **Sudo ./manager-linux.run**

Generar la documentación de manera automática: phpdoc run –d . –t doc

La documentación se generará y almacenará en la carpeta doc. <http://localhost/banco/doc>

## 2.3.-Configuración de phpDocumentor

Las tres opciones esenciales para generar la documentación son:

-**d.** Especifica el directorio o directorios del proyecto que quieres documentar.

**-f.** Especifica el fichero o fcheros que quieres documentar.

**-t.** Especfica la localización donde quieres que se almacenen los archivos generados. Este parámetro es opcional. Si no se especifica, los archivos se almacenarán en la carpeta output.

Otras opciones de configuración son:

**--ignore.** Se pueden excluir ciertos ficheros del proceso de generación de documentación.

**--hidden, --gnore-symlinks.** Por defecto phpDocumentor ignora los archivos ocultos y los enlaces simbólicos. Si se quieren incluir se puede escribir la opción --hidden=off y --no-ignore-symlinks.

**--template.** Se pueden usar varias plantillas HTML para generar la documentación.

**--title.** Esta opción cambia el título en la pestaña del navegador.

**--encoding.** Se asume que los caracteres del proyecto están codificados con UTF-8 pero se puede cambiar si no fuera el caso.

**--visibility.** Se pueden limitar los elementos que aparecen en la documentación dependiendo del modificador de acceso (public, protected y private) con el que hayan sido definidos.

**--ignore-tags.** Se pueden omitir ciertas etiquetas durante la generación de la documentación.

**--config.** Sirve para indicar el fichero de configuración utilizado en el proceso de generación de documentación.

phpDocumentor tiene un fichero de configuración:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <phpdocumentor  configVersion="3"  xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="https://www.phpdoc.org"  noNamespaceSchemaLocation="https://raw.githubusercontent.com/phpDocumentor/phpDocum>  <paths>  <output>build/api</output>  <cache>build/cache</cache>  </paths>  <version number="3.0.0">  <api>  <source dsn=".">  <path>src</path>  </source>  </api>  </version>  </phpdocumentor> |

## 3.-JavaDoc

Es una utilidad de Sun Microsystems empleado para generar APIs (Aplication Programing Interface) en formato HTML de un archivo de código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java, la mayoría de los IDEs los generan automáticamente.

**Javadoc** es un programa, que recoge los comentarios que se colocan en el código con marcas especiales y construye un archivo HTML con clases, métodos y la documentación que corresponde. Este HTML tiene el formato de toda la documentación estándar de Java provista por Sun.

Se utiliza /\*\* al comienzo y al final \*/, dentro de estos comentarios se puede escribir código HTML y etiquetas @.

Estas etiquetas permiten generar una API completa a partir del código fuente con los comentarios.

Son sensibles a mayúsculas-minúsculas, se sitúa siempre al principio de una línea o sólo precedida por espacio(s) y arterisco(s) para que la herramienta Javadoc la interprete como tal. Si no se hace así las interpretará como texto normal.

Dos tipos:

* Etiquetas de bloque: sólo se pueden utilizar en la sección de etiquetas que sigue a la descripción principal. @etiqueta
* Etiquetas inline: se pueden utilizar tanto en la descripción principal como en la sección de etiquetas. {@tag}

## 3.1.- Instalación de Javadoc.

* Instalación de Ubuntu 20.04 o superior
* Instalación de Eclipse o NetBeans
* Eclipse
  + Apt install default-jre
  + Snap install --classic eclipse
* NetBeans
  + Apt install default-jdk
* wget <https://downloads.apache.org/netbeans/netbeans/17/netbeans-17-bin.zip>
* unzip netbeans-17-bin.zip
* mv netbeans/ /opt/
* export PATH="$PATH:/opt/netbeans/bin/"
* $ source ~/.bashrc
* /usr/share/applications/netbeans.desktop (acceso directo)

|  |
| --- |
| [Desktop Entry]  Name=Netbeans IDE  Comment=Netbeans IDE  Type=Application  Encoding=UTF-8  Exec=/opt/netbeans/bin/netbeans  Icon=/opt/netbeans/nb/netbeans.png  Categories=GNOME;Application;Development;  Terminal=false  StartupNotify=true |

Tanto Eclipse como NetBeans disponen entre sus opciones la de generar javadoc y mediante diversas ventanas que ofrecen se pueden seleccionar las opciones para javadoc. Pero no sólo eso, sino también ofrecen el completado de código javadoc.

Menú > Project > Generate Javadoc > Seleccionar proyecto y ruta donde se guardará. Documentación generada.

JDK instalado > javadoc ejemplo.java

## 3.2.-Funcionamiento de Javadoc

Los comentarios JavaDoc están destinados a describir, principalmente, clases y métodos.

Como están pensados para que otro programador los lea y utilice la clase (o método) correspondiente, se decidió fijar, al menos parcialmente, **un formato común,** de forma que los comentarios escritos por un programador resultaran legibles por otro. Para ello los comentarios JavaDoc deben incluir unos indicadores especiales, que comienzan siempre por '@' y se suelen colocar al comienzo de línea.

En JavaDoc, la descripción de la clase o del método no va precedida de ningún indicador. Se usan indicadores para el número de versión (@version), el autor (@author) y otros. Es importante observar que los indicadores no son obligatorios; por ejemplo, en un método sin parámetros no se incluye obviamente el indicador @param. También puede darse que un comentario incluya un indicador más de una vez, por ejemplo varios indicadores @param porque el método tiene varios parámetros. Resumiendo, los indicadores más usuales:

**@author nombreDelAutor descripción.** Indica quién escribió el código al que se refiere el comentario. Si son varias personas se escriben los nombres separados por comas o se repite el indicador, según se prefiera.

**@version númeroVersión descripción.** Si se quiere indicar la versión. Normalmente se usa para clases, pero en ocasiones también para métodos.

**@param nombreParámetro descripción.** Para describir un parámetro de un método.

**@return descripción.** Describe el valor de salida de un método.

**@see nombre descripción.** Cuando el trozo de código comentado se encuentra relacionada con otra clase o método, cuyo nombre se indica en nombre.

**@throws nombreClaseExcepción descripción.** Cuando un método puede lanzar una excepción ("romperse" si se da alguna circunstancia) se indica así.

**@deprecated descripción.** Indica que el método (es más raro encontrarlos para una clase) ya no se usa y se ha sustituido por otro.

* Tanto Eclipse como NetBeans disponen de opciones para generar javadoc y completado de código.
* Menú > Project > Generate Javadoc (seleccionar proyecto y ruta de generación de la documentación)
* Consultar el Html generado.
* También se puede ir al proyecto y ejecutar directamente javadoc: **javadoc -d doc src/banco/\***

## 3.3.-Creación y uso de plantillas de código.

La documentación del API de Java ha sido creada de este modo. Esto hace el trabajo de documentar el código de nuevas clases Java sea trivial.

Las plantillas:

* Son sugerencias de código asociadas a palabras clave.
* En Apache Netbeans se hallan definidas en **Tools > Options > Editor > Code Templates.**
* Es aconsejable examinar todas, ya que pueden ahorrar mucho trabajo.
* Muchas de ellas utilizan nombres similares a las construcciones Java que encapsulan (try, for, while, if,...).
* Podemos definir y crear nuestras propias plantillas.
* Además existen plantillas Javadoc predefinidas.

Una plantilla se compone de:

* un nombre,
* una descripción,
* un contexto en función del lenguaje (en java, si estamos en el código, en el javadoc,...) y
* un pattern, que es el código de la plantilla. Dentro del código de la plantilla podemos usar texto fijo o una serie de variables predefinidas, por ejemplo:
  + **${cursor}:** posición en la que se establecerá el cursor de texto tras desplegar el código de la plantilla.
  + **${enclosing\_type}:** tipo de la clase en la que nos encontramos.
  + **${enclosing\_method}:** nombre del método en el que nos encontramos.
  + **${year}:** año en curso.
  + **${time}:** hora en curso.

Estas **plantillas** se mostrarán como sugerencias en el código tras comenzar a escribir su nombre y pulsar **CTRL+ espacio**. Podemos crearnos nuestras propias plantillas, además de modificar las existentes. Para ello no tenemos más que añadir una nueva desde la opción de "**Templates**", asignarle un nombre, descripción y elegir el código que queremos que se muestre al seleccionar la misma.

## 4.-Sistemas de control de versiones.

Los sistemas de control de versiones, que nos ayudan a guardar las distintas versiones de los fuentes.

Con un SCV hay un directorio, controlado por esta herramienta, donde se van guardando los fuentes de nuestro proyecto con todas sus versiones. Usando esta herramienta, nosotros sacamos una copia de los fuentes en un directorio de trabajo, ahí hacemos todos los cambios que queramos y, cuando funcionen, le decimos al sistema de control de versiones que nos guarde la nueva versión. El sistema de control de versiones suele pedirnos que metamos un comentario cada vez que queremos guardar fuentes nuevos o modificados.

Podemos obtener fácilmente cualquiera de las versiones de nuestros fuentes, ver los comentarios que pusimos en su momento e, incluso, comparar distintas versiones de un mismo fuente para ver qué líneas hemos modificado.

Aunque los sistemas de control de versiones se hacen imprescindibles en proyectos de cierta envergadura y con varios desarrolladores, de forma que puedan mantener un sitio común con las versiones de los fuentes a través de un sistema de control de versiones, también puede ser útil para un único desarrollador en su casa, de forma que siempre tendrá todas las versiones de su programa controladas.

Los sistemas de control de versiones son programas que permiten gestionar un repositorio de archivos y sus distintas versiones; utilizan una arquitectura cliente-servidor en donde el servidor guarda la(s) versión(es) actual(es) del proyecto y su historia. Sirven para mantener distintas versiones de un fichero, normalmente código fuente, documentación o ficheros de configuración.

## 4.1.- Conceptos básicos de sistemas de control de versiones.

* **Revisión**: es una visión estática en el tiempo del estado de un grupo de archivos y directorios. Posee una etiqueta que la identifica. Suele tener asociado metadatos:
  + Identidad de quién hizo las modificaciones.
  + Fecha y hora en la cual se almacenaron los cambios.
  + Razón para los cambios.
  + De qué revisión y/o rama se deriva la revisión.
  + Palabras o términos clave asociados a la revisión.
* **Copia de trabajo:** Árbol de trabajo, es el conjunto de directorios y archivos controlados por el sistema de control de versiones, y que se encuentran en edición activa. Está asociado a una rama de trabajo concreta.
* **Rama de trabajo:** es un conjunto ordenado de revisiones. La revisión más reciente se denomina principal (main). Las ramas se pueden separar y juntar según como sea necesario, formando un grafo de revisión.
* **Repositorio**: lugar en donde se almacenan las revisiones. Físicamente puede ser un archivo, colección de archivos, base de datos, etc. Y puede estar almacenado en local o en remoto.
* **Conflicto**: ocurre cuando varias personas han hecho cambios contradictorios en un mismo documento/s. El proceso de solucionar un conflicto se denomina resolución.
* **Cambio**: modificación en un archivo bajo control de revisiones. Cuando se unen los cambios en un archivo o varios, generando una revisión unificada, se dice que se ha hecho una combinación o integración.
* **Parche**: lista de cambios generada al comparar revisiones y que puede usarse para reproducir automáticamente las modificaciones hechas en el código.

Con un scv se consigue mantener un repositorio con la información actualizada. La forma habitual de trabajar consiste en mantener una copa en local y modificarla. Después actualizarla en el repositorio.

Ventaja, no es necesario el acceso continuo al repositorio.

## 4.2.- Procedimiento de uso habitual de un sistema de control de versiones.

Ciclo de operaciones:

* Descarga de ficheros inicial (Checkout):
  + El primer paso es bajarse los ficheros del repositorio.
  + El Checkout sólo se hace la primera vez que se usan esos ficheros.
* Ciclo de trabajo habitual:
  + Modificación de los ficheros, para aplicar los cambios oportunos como resultado de la aportación de cada usuario.
  + Actualización de ficheros en local (update): los ficheros son modificados en local y luego se sincronizan con los ficheros existentes en el repositorio.
  + Resolución de conflictos: se informa del conflicto y los usuarios implicados en la manipulación del código afectado por el conflicto, los encargados de solucionarlo.
  + Actualización de ficheros en repositorio (commit): consiste en la modificación de los ficheros en el repositorio; el scv comprueba que las versiones que se suben estén actualizadas.

Los sistemas de control de versiones permiten las siguientes funciones:

* Varios clientes pueden sacar copias del proyecto al mismo tiempo.
* Realizar cambios a los ficheros manteniendo un histórico de los cambios:
  + Deshacer los cambios hechos en un momento dado.
  + Recuperar versiones pasadas.
  + Ver históricos de cambios y comentarios.
* Los clientes pueden también comparar diferentes versiones de archivos.
* Unir cambios realizados por diferentes usuarios sobre los mismos ficheros.
* Sacar una "foto" histórica del proyecto tal como se encontraba en un momento determinado.
* Actualizar una copia local con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.
* Mantener distintas ramas de un proyecto.

Los scv ofrecen establecer algún método de autorización, es decir, la posibilidad por la cual a ciertas personas se les permite o no realizar cambios en áreas específicas del repositorio.

## 4.3.-Sistemas de control de versiones centralizados y distribuidos.

Sistema de control de versiones locales, contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos.

Los **rcs**, funciona básicamente guardando conjuntos de parches (es decir, la diferencias entre archivos) de una versión a otra en un formato especial en disco, puede entonces recrear cómo era un archivo en cualquier momento sumando los distintos parches.

Los sistemas de control de versiones centralizados (CVCS), como CVS, Subversión y Perforce, tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos de ese lugar central.

Ventajas:

* Todo el mundo sabe hasta cierto punto en qué está trabajando el resto de gente en el proyecto.
* Los administradores tienen control detallado de qué puede hacer cada uno, y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Desventaja:

* Un servidor centralizado con todas la problemática que acarrea.

Sistema de control de versiones distribuido (DVCS): git, mercurial, bazaar, darcs, los clientes no sólo descargan la última instantánea de los archivos sino que replican completamente el repositorio (copia completa de todos los datos).

Es más, muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que se puede colaborar con distintos grupos de gente de maneras distintas simultáneamente dentro del mismo proyecto. Esto permite establecer varios tipos de flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

4.4.- Git como sistema de control de versiones.