DESPLIEGUE DE APLICACIONES WEB

CFGS. Diseño de Aplicaciones Web (DAW). 18-19

IES FRANCESC DE BORJA MOLL

TEMA 6. DOCUMENTACIÓN Y CONTROL DE VERSIONES.

ÍNDICE DE CONTENIDOS:

1. DOCUMENTACIÓN.

Para desarrollar un proyecto informático necesario generar muchos tipos diferentes de documentación a cada fase del proyecto. Los elementos que se documentarán dependen de la metodología utilizada para planificar el proyecto. Por ejemplo, si se utiliza una metodología en cascada la documentación debe ser exhaustiva, mientras que en el caso de aplicar metodologías ágiles el número de elementos es mucho menor.

Entre los diferentes tipos de documentación hay que destacar la documentación del código, ya que los encargados de generar esta documentación son los programadores y ésta se incluye en forma de comentarios al código fuente de los programas, aunque es posible extraerla (en formato HTML , por ejemplo) para consultarla externamente.

Por otra parte, como habitualmente en el desarrollo de un proyecto informático participa más de una persona, también es muy habitual encontrar la documentación como lugar wiki. A diferencia de la documentación del código, que es generado automáticamente a partir de los comentarios, la documentación de un sitio wiki es escrita por los diferentes miembros del proyecto.

Hay que diferenciar entre los formatos de documentación y las herramientas para generar la documentación externa, ya que un mismo generador puede aceptar uno o más formatos diferentes, pero un formato no puede mezclarse con los demás.

También hay que destacar que la utilización de un formato u otro no depende sólo del lenguaje, sino que diferentes bibliotecas o entornos de trabajo pueden utilizar formatos diferentes.

1.1. Ventajas de documentar el código.

La tarea de documentar el código de la aplicación utilizando comentarios no sólo sirve para que pueda ser consultado por terceros, sino para vosotros mismos, ya que a lo largo de la vida de un proyecto hay que hacer cambios, mejoras y mantenimiento.

Por ejemplo, tal vez tiene que encargar de arreglar una función que voy implementar hace más de seis meses, y durante ese tiempo seguramente ha estado trabajando en otros proyectos. Si el código está correctamente documentado, no te costará nada entender cómo funciona, pero, si no hay ningún tipo de documentación, deberá depurar e inspeccionar el código para entender de nuevo su funcionamiento. Otro caso muy habitual es tener que trabajar con código de terceros. Si este código no está documentado, el tiempo que tardará en entender cómo funciona y cómo solucionar el problema o aplicar los cambios aumentará enormemente.

En el caso de trabajar en proyectos de software libre esta tarea es aún más importante, ya que el código se utiliza y se modifica no sólo por su equipo de desarrollo, sino por cualquier persona interesada. En estos casos hay que prestar especial atención a la documentación de los componentes públicos de la

aplicación porque son los elementos a los que tendrán acceso otros desarrolladores.

En cualquier caso, hay que tener en cuenta que, al igual que en el caso de los comentarios, una documentación incorrecta o desactualizada es peor que una documentación inexistente, ya que los usuarios confían en la veracidad.

1.2. Buenas prácticas.

Hay que tener en cuenta que no se recomienda utilizar comentarios en el código cuando no forman parte de la documentación, que no son mantenidos y pueden llegar a confundir otros desarrolladores.

Mucho peor que no encontrar ningún comentario respecto a un fragmento de código es encontrar un comentario desfasado o que contenga información errónea. En muchos casos estos comentarios (que nadie recuerda quien los añadió ni qué función tienen) se van arrastrando a lo largo de los años a medida que el código se modifica.

En lugar de ello, lo que se recomienda es utilizar buenas prácticas para nombrar los elementos del software (clases, funciones, variables, etc.) y escribir el código de manera comprensible. Por ejemplo, en caso de utilizar nombres de variables crípticos, hay que añadir comentarios para explicar qué hace un fragmento de código. Por otra parte, la documentación del código permite paliar algunas de las deficiencias de los lenguajes o entornos de trabajos, y esto ayuda a comunicar la intencionalidad del código a otros usuarios. Por ejemplo, en JavaScript no existe el concepto de ámbito privado, público o protegido, pero se puede etiquetar el código indicando cuál debería ser su ámbito:

```
// Contiene el precio total de una línea de pedido
// que consiste en la multiplicación del precio
// unitario por la cantidad multiplicado por
// impuestos añadiendo el coste de envío.
float t = p * q * 0.21f + e;
```

En cambio, si se da un nombre claro a las variables, se entiende claramente que hace el código:

```
const float IVA = 0.21f;
float totalLinia = preuUnitari * cantidad * IVA + envío;
```

Como se puede ver, en el segundo caso no hay ningún comentario: se puede deducir correctamente el funcionamiento y la intencionalidad del código.

En caso de que el código sea demasiado complicado y que a pesar de utilizar nombres adecuados no se entienda, lo que hay que hacer es reescribir el código de una manera más simple. Recuerde que esto no sólo ayuda otros desarrolladores que tengan que trabajar con el código, sino a vosotros mismos en el futuro, si la ha de mantener.

1.3. Documentación con Javadoc.

Javadoc es un generador de documentación para el lenguaje Java. Se utiliza desde la primera versión de Java y se actualiza con cada nueva versión. Esta herramienta se encuentra incluida en el Java Development Kit (JDK), de modo que si desarrollas aplicaciones con Java es muy probable que ya esté instalada en el sistema.

El formato utilizado para Javadoc es el estándar de facto. Por este motivo se encuentran muchos generadores con un formato similar, como **JSDoc y phpDocumentor**. Además, se puede acceder al generador de documentación directamente desde las IDE más populares, como son IntelliJ, Eclipse o NetBeans.

Hay que tener en cuenta que los comentarios son descartados cuando se compila el código fuente, por lo que documentar el código no afecta al rendimiento del programa.
Para añadir información adicional a la documentación, Javadoc utiliza un sistema de etiquetas prefijado con el símbolo de la arroba. Por ejemplo: @author, @param o @return. Cabe destacar que el número de etiquetas de Javadoc es mucho más reducido que el de otros generadores como JSDoc, ya que, como Java es un lenguaje fuertemente tipado, la mayoría de las restricciones se encuentran definidas por el mismo código y no es necesario indicar -las. Por ejemplo: la privacidad de las propiedades y métodos, cuando se trata de una clase y cuando se trata de una función.

A continuación puede encontrar una lista de las etiquetas más utilizadas en Javadoc:

```
@author nombre: indica el autor del código.
@deprecated texto: indica que este método o clase no se debe utilizar y, opcionalmente, el motivo.
@exception clase descripción, @throw clase descripción: son sinónimas e indican que este método puede lanzar una excepción.
@param nombre descripción: añade información sobre un parámetro.
@return descripción: añade información sobre el valor de retorno de un método.
@link paquet.classe # miembro etiqueta: inserta un enlace que apunta a otro elemento de la documentación.
@see referencia: indica que este elemento está relacionado con otro. Pueden añadirse múltiples etiquetas @see en un mismo comentario, cada una en una línea.
@since texto: indica en qué versión del software se añadió esta clase o método.
@version texto: indica la versión.
```

Para más información:
https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

1.4. Documentación con phpDocumentator.

phpDocumentor es un generador de documentación de código abierto escrito en PHP. Automáticamente analiza el código fuente PHP y produce la API de lectura y documentación del código fuente en una variedad de formatos. phpDocumentor genera la documentación en base al estándar formal PHPDoc.

Es compatible con la documentación del código orientado a objetos y programación procedimental, además es capaz de crear documentos HTML, PDF, CHM y formatos Docbook. Se puede utilizar desde la línea de comandos o mediante una interfaz web. Tiene soporte para la vinculación entre la documentación, la incorporación de documentos a nivel de usuario como tutoriales, y la creación de código fuente resaltado con referencias cruzadas a la documentación en general de PHP.
phpDocumentor es capaz de analizar toda la sintaxis de PHP y apoya PHP4 y PHP5. Se trata de un proyecto de código abierto y se distribuye bajo la licencia GPL.

Existen tres tipos de documentaciones:

- **Interfaz** (para los usuarios del código): qué hace, como se utiliza, que devuelve, … Pero no cómo lo hace.
- **Implementación** (para editores del código): cómo funciona internamente, que algoritmos utiliza, …
- **Toma de decisiones** (para editores y responsables de desarrollo): por qué se ha implementado de una forma o otra (razones de rendimiento, recursos, …).

La documentación de la implementación reside **dentro del código**, y la de la interfaz ha de ser **un documento**. Pues **PHPDocumentor se encarga de convertir** la documentación que existe en el código, a documentación legible desde un documento como una página web. De esta forma, cada vez que se aplique un cambio en el código, esto repercutirá en la documentación de la interfaz, **manteniéndola actualizada constantemente**.

Para conseguir esto, la documentación en el código tiene que seguir unos estándares. Se escriben unos bloques de documentación llamados DocBlock, que siguen una estructura y pueden tener una serie de marcas para ayudar a PHPDocumentor a entender mejor la documentación.

Se puede comentar cualquier archivo PHP (.php, .php5, .phtml), y dentro de ellos se pueden documentar: clases, variables, defines, funciones, variables globales y llamadas a otros ficheros. En los DocBlock, unas de las **principales marcas** que se pueden utilizar son:

```
@access: Si @access es 'private' no se genera documentación para el
elemento (a menos que se indique explícitamente). Muy interesante si
sólo se desea generar documentación sobre la interfaz (métodos públicos)
pero no sobre la implementación (métodos privados).
@author: Autor del código.
@copyright: Información sobre derechos.
```

@deprecated: Para indicar que el elemento no debería utilizarse, ya que en futuras versiones podría no estar disponible.
@example: Permite especificar la ruta hasta un fichero con código PHP. phpDocumentor se encarga de mostrar el código resaltado (syntax-highlighted).
@ignore: Evita que phpDocumentor documente un determinado elemento.
@internal: Para incluir información que no debería aparecer en la documentación pública, pero sí puede estar disponible como documentación interna para desarrolladores.
@link: Para incluir un enlace (http://…) a un determinado recurso.
@see: Se utiliza para crear enlaces internos (enlaces a la documentación de un elemento).
@since: Permite indicar que el elemento está disponible desde una determinada versión del paquete o distribución.
@version: Versión actual del elemento

Marcas para **funciones**:

@global: Permite especificar el uso de variables globales dentro de la función.
@param: Parámetros que recibe la función. Formato: *@param tipo $nombre_var comentario*
@return: Valor devuelto por la función. Formato: *@return tipo comentario*

Marca para **variables**:

@var: Documenta los atributos de la clase. Formato: *@var tipo comentario*

Para más información:
https://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_phpDocumentor.howto.pkg.html

2. CONTROL DE VERSIONES.

El **control de versiones** permite mantener un registro de cambios en un conjunto de documentos a lo largo del tiempo. Este tipo de control no se limita sólo al desarrollo de hardware, sino que se ha aplicado durante muchos años en la gestión de documentos en papel, donde se identifica cada versión del documento con un número, la fecha y hora en la que se ha generado la revisión y el nombre de la persona que ha hecho los cambios.

En ingeniería del software el control de versiones o revisiones es la práctica que proporciona control sobre los cambios en el código fuente de un programa. Habitualmente este control se lleva a cabo utilizando herramientas especializadas como son CVS, Mercurial, Git, etc.

Estas herramientas tradicionalmente se ejecutan desde la línea de comandos, pero la mayoría de sistemas de control de versiones modernos proporcionan también una interfaz gráfica y pueden integrarse con los entornos de desarrollo integrados (Integrated Development Environment o IDE en inglés) más populares.

Hay que tener en cuenta que los sistemas de control de versiones también se encuentran integrados en otro tipo de software, como por ejemplo los documentos colaborativos (Google Docs) o las wikis (Wikipedia y DokuWili), que implementan sus propios sistemas de control de versiones para poder revisar los cambios y restaurar versiones anteriores.

Disponer de un sistema de control de versiones resulta imprescindible a la hora de afrontar la implementación de cualquier aplicación mínimamente compleja, ya que proporcionan a los desarrolladores la opción de deshacer cambios y volver a versiones anteriores del desarrollo fácilmente.

Estos sistemas están pensados para trabajar principalmente con ficheros de código, es decir, archivos de texto y otros tipos de contenidos como imágenes, vídeos o archivos de audio normalmente no se guardan utilizando el mismo sistema (el conjunto de cambios en cada línea) sino que se deben guardar enteros.

Esto en determinados tipos de proyectos puede presentar un problema, ya que los requerimientos de espacio por el sistema de control de versiones pueden ser muy grandes y pueden producirse errores si no se tiene en cuenta.

2.1. Repositorio

El lugar donde se guardan los conjuntos de cambios se denomina repositorio. Normalmente se trata de un directorio donde se encuentran todos los archivos del proyecto y los datos con el historial de cambios, lo que permite hacer una copia de estos archivos en cualquier momento concreto, determinado por su número de versión. Cada vez que se añaden cambios en el repositorio se crea una nueva entrada en el historial de cambios donde se incluye el número de versión y a partir de este número es posible restaurar este estado.

Hay que distinguir entre los repositorios locales y los repositorios remotos. Según la herramienta utilizada, un repositorio remoto se encuentra en un servidor independiente o se trata de un depósito local de otro equipo. En cualquiera de los dos casos el concepto más importante es que los repositorios pueden sincronizarse de manera que diferentes usuarios pueden trabajar en el mismo proyecto y los mismos ficheros simultáneamente.

2.2. Tronco y ramas.

La estructura de un sistema de control de versiones se puede interpretar como un árbol, **donde el tronco o rama principal es donde se suben los cambios realizados en la aplicación y pueden crearse ramas para trabajar en nuevas características o soluciones de errores que más adelante se añadirán al tronco**. De esta manera se puede trabajar de forma segura en una rama sin que los cambios del trabajo en progreso afecten la rama principal.

Hay varios casos en que una rama no vuelve a fusionarse con el tronco. Por ejemplo, se podría crear una rama para comprobar si una funcionalidad es viable o si una implementación diferente de una funcionalidad existente mejora el rendimiento y, según el resultado, hacer la fusión con la rama principal o descartarla.

En otros casos puede crearse una rama para mantener los cambios de una versión anterior (por ejemplo, la versión 1 del software) mientras que el tronco se trabaja con una nueva versión (por ejemplo, la versión 2). De este modo, aunque la rama principal se trabaja con una nueva versión, si se detectan errores en la versión 1 pueden solucionarse y subir estos cambios a la rama correspondiente a la versión 1 (para distribuirlos a los clientes que todavía utilicen esta versión).

No hay límite en el número de ramas que pueden crearse.

2.3. Tipos de control de versiones.

Según su arquitectura, los sistemas de control de versiones pueden dividirse en los siguientes tipos:

- **Sistemas centralizados**: usan una arquitectura cliente-servidor donde todos los clientes conectan con el servidor para subir o bajar los cambios, y el equipo local no guarda ningún historial de los cambios.
- **Sistemas distribuidos o descentralizados**: cada usuario tiene una copia completa de todos los archivos, el historial de cambios y las subidas y bajadas se realizan entre los repositorios de cada usuario para sincronizarse entre ellos.

Los **sistemas centralizados presentan la ventaja de reducir el peso del proyecto en los clientes**, ya que éstos sólo contienen una copia de los ficheros y no de los cambios anteriores. Por el contrario, si falla el servidor ninguno de los

clientes podrá ni bajar los cambios ni subir los nuevos. En el peor de los casos, si se pierden los datos del servidor, se perdería toda la información del repositorio (si no hay copia de seguridad).

En el caso de los sistemas distribuidos, si un cliente falla no hay ningún problema, porque puede recuperarse el repositorio completo junto con el historial de cambios de cualquier otro equipo, por lo que sólo se perdería la información no subida desde el cliente que ha fallado. El inconveniente es que como no hay un servidor central deben sincronizar todos los repositorios entre ellos o seleccionar un repositorio como central y hacer que todos los repositorios utilicen el mismo para subir y bajar los cambios.

Actualmente, la opción más popular es una mezcla entre los dos sistemas: utilizar un servidor central como GitHub, ya que se trata de una plataforma muy segura, y utilizar a los clientes un sistema distribuido (Git).

Otra característica importante para diferenciar entre tipos de controles de versiones es el sistema de concurrencia utilizado:

- **Fusión (merge):** cuando se produce un conflicto porque se han hecho cambios sobre un fichero que ha sido modificado, se resuelve fusionando los archivos (automáticamente o manualmente).
- **Bloqueo (lock):** cuando un usuario modifica un archivo, éste se bloquea de manera que ningún otro usuario puede modificarlo.

Hay que tener en cuenta que algunos de los sistemas de control de versiones ofrecen todas las opciones, es decir, pueden utilizarse como sistema centralizado o distribuido y permitir el uso de los modelos de concurrencia de fusión y de bloqueo.

2.4. Terminología.

Hay que tener en cuenta que no todos los sistemas de control de versiones utilizan los mismos términos para referirse a los mismos conceptos. Por este motivo es importante familiarizarse con la terminología. A continuación, una lista con el nombre en catalán de los términos más comunes y el nombre o nombres en inglés relacionados:

- **Repositorio (repository o depot):** hace referencia al lugar donde se guardan los archivos actuales y el histórico de cambios.
- **Tronco (trunk o master):** es la rama principal de un repositorio de la que salen el resto de ramas.
- **Rama (branch):** es una bifurcación del tronco o rama maestra de la aplicación que contiene una versión independiente de la aplicación y en la que pueden aplicarse cambios sin que afecten ni el tronco ni otras ramas.
- **Cabecera (head o tip):** hace referencia a la versión más reciente de una determinada rama o del tronco. El tronco y cada rama tienen su propia cabeza, pero para referirse al jefe del tronco a veces se utiliza el término HEAD, en mayúsculas.
- **Copia de trabajo (working copy):** hace referencia a la copia local de los archivos que se han copiado del repositorio, que es sobre la que se hacen

los cambios (es decir, se trabaja, de ahí el nombre) antes de añadir estos cambios al repositorio.

- **Bifurcación (fork):** consiste en crear un nuevo repositorio a partir de otro. Este nuevo repositorio, al contrario que en el caso de la clonación, no está ligado al repositorio original y se trata como un repositorio diferente.
- **Clonar (clone):** consiste en crear un nuevo repositorio que es una copia idéntica de otro, ya que contiene las mismas revisiones.
- **Subir (commit o check in):** es añadir los cambios locales en el repositorio. Cabe destacar que no los envía al servidor; los cambios quedan almacenados en el repositorio local que se debe sincronizar.
- **Bajar (check out):** es copiar en el área de trabajo local una versión desde un repositorio local, un repositorio remoto o una rama diferente.
- **Pull:** es la acción que copia los cambios de un repositorio (habitualmente remoto) en el depósito local. Esta acción puede provocar conflictos.
- **Push o fetch:** son acciones utilizadas para añadir los cambios del repositorio local a otro repositorio (habitualmente remoto). Esta acción puede provocar conflictos.
- **Cambio (change o diff):** representa una modificación concreta de un documento bajo el control de versiones.
- **Sincronización (update o sync):** es la acción de combinar los cambios hechos al repositorio con la copia de trabajo local.
- **Conflicto (conflict):** se produce cuando se intentan añadir cambios a un fichero que ha sido modificado previamente por otro usuario. Antes de poder combinar los cambios con el repositorio deberá resolver el conflicto.
- **Bloqueo (lock):** algunos sistemas de control de versiones en lugar de utilizar el sistema de fusiones lo que hacen es bloquear los archivos en uso, por lo que sólo puede haber un solo usuario modificando un fichero en un momento dado.
- **Fusionar (merge o integration):** es la acción que se produce cuando se quieren combinar los cambios de un repositorio local con un remoto y se detectan cambios en el mismo archivo en ambos repositorios y se produce un conflicto. Para resolver este conflicto se deben fusionar los cambios antes de poder actualizar los repositorios. Esta fusión puede consistir en descartar los cambios de uno de los dos repositorios o editar el código para incluir los cambios del archivo en ambos lados. Cabe destacar que es posible que un mismo fichero presente cambios en muchos puntos diferentes que deben ser resueltos para poder dar la fusión por finalizada.
- **Versión (version o revision):** es el conjunto de cambios en un momento concreto del tiempo. Se crea una versión cada vez que se añaden cambios a un repositorio.
- **Etiqueta (tag, label o baseline):** permite añadir una etiqueta a una subida para poder identificar esta subida concreta de una forma más comprensible. Por ejemplo, se puede etiquetar la primera versión de un software (1.0) o una versión en la que se ha solucionado un error importante.
- **Volver a la versión anterior (revert):** descarta todos los cambios producidos en la copia de trabajo desde la última subida al depósito local.

2.5. **Software de control de versiones.**

Hay muchos sistemas de control de versiones, tanto gratuitos como de pago. Entre las opciones más populares, por orden de popularidad, son:

- **Git:** es el software más popular con diferencia. Se trata de un sistema distribuido, utiliza el modelo de concurrencia de fusión y es gratuito. Sus principales órdenes son:

```
git fetch:
Descarga los cambios realizados en el repositorio remoto.
git merge <nombre_rama>:
Impacta en la rama en la que te encuentras parado, los cambios
realizados en la rama "nombre_rama".
git pull:
Unifica los comandos fetch y merge en un único comando.
git commit -am "<mensaje>":
Confirma los cambios realizados. El "mensaje" generalmente se usa para
asociar al commit una breve descripción de los cambios realizados.
git push origin <nombre_rama>:
Sube la rama "nombre_rama" al servidor remoto.
git status:
Muestra el estado actual de la rama, como los cambios que hay sin
commitear.
git add <nombre_archivo>:
Comienza a trackear el archivo "nombre_archivo".
git checkout -b <nombre_rama_nueva>:
Crea una rama a partir de la que te encuentres parado con el nombre
"nombre_rama_nueva", y luego salta sobre la rama nueva, por lo que
quedas parado en esta última.
git checkout -t origin/<nombre_rama>:
Si existe una rama remota de nombre "nombre_rama", al ejecutar este
comando se crea una rama local con el nombre "nombre_rama" para hacer un
seguimiento de la rama remota con el mismo nombre.
git branch:
Lista todas las ramas locales.
git branch -a:
Lista todas las ramas locales y remotas.
git branch -d <nombre_rama>:
Elimina la rama local con el nombre "nombre_rama".
git push origin <nombre_rama>:
Commitea los cambios desde el branch local origin al branch
"nombre_rama".
git remote prune origin:
Actualiza tu repositorio remoto en caso que algún otro desarrollador
haya eliminado alguna rama remota.
git reset --hard HEAD:
Elimina los cambios realizados que aún no se hayan hecho commit.
git revert <hash_commit>:
Revierte el commit realizado, identificado por el "hash_commit".
```

- **Subversion (SVN):** Apache Subversion (abreviado frecuentemente como SVN, por el comando svn) es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD.

Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco. SVN permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo

haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras. A cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración

- **Team Foundation Server (TFS):** es un software desarrollado por Microsoft que puede utilizar las arquitecturas centralizadas o distribuidas y hacer el modelo de fusión o bloqueo. Es gratuito para equipos pequeños y proyectos de código libre, mientras que en otros casos hay que pagar una suscripción.
- **Mercurial**: se trata de un sistema distribuido, utiliza el modelo de fusión y es gratuito.