UT 4. Optimización y documentación. Control de versiones

1. Repositorio. Herramientas de control de versiones. Uso integrado en el entorno de desarrollo:

Con el término versión, se hace referencia a la evolución de un único elemento, o de cada elemento por separado, dentro de un sistema en desarrollo
Siempre que se está realizando una labor, sea del tipo que sea, es importante saber en cada momento de que estamos tratando, qué hemos realizado y qué nos queda por realizar.
En el caso del desarrollo de software ocurre exactamente lo mismo. Cuando estamos desarrollando software, el código fuente está cambiando continuamente, siendo esta particularidad vital. Esto hace que, en el desarrollo de software actual, sea de vital importancia que haya sistemas de control de versiones.

Las ventajas de utilizar un sistema de control de versiones son múltiples:
Un sistema de control de versiones bien diseñado **facilita al equipo de desarrollo su labor,** permitiendo que varios desarrolladores trabajen en el mismo proyecto (incluso sobre los mismos archivos) de forma simultánea, sin que sé que pisen unos a otros.
Las herramientas de control de versiones proveen de **un sitio central donde almacenar el código fuente de la aplicación**, así como el historial de cambios realizados a lo largo de la vida del proyecto.
También permite a los desarrolladores **volver a una versión estable previa** del código fuente si es necesario.

Una versión, desde el punto de vista de la evolución, se define como la forma particular de un objeto en un instante o contexto dado.
Se denomina revisión, cuando se refiere a la evolución en el tiempo.
Pueden coexistir varias versiones alternativas en un instante dado y hay que disponer de un método, para designar las diferentes versiones de manera sistemática u organizada.

En los entornos de desarrollo modernos, los sistemas de control de versiones son una parte fundamental, que van a permitir construir técnicas más sofisticadas como la Integración Continua.

En los proyectos Java, existen dos sistemas de control de versiones de código abierto, CVS(*Concurrent Versions System)* y Subversion. La herramienta CVS es una herramienta de código abierto que es usada por gran cantidad de organizaciones. Subversion es el sucesor natural de CVS, ya que se adapta mejor que CVS a las modernas prácticas de desarrollo de software.

1.1. Estructura de herramientas de control de versiones.

Las herramientas de control de versiones suelen estar formadas por un conjunto de elementos, sobre los cuales, se pueden ejecutar órdenes e intercambiar datos entre ellos. Como ejemplo, vamos a analizar ha herramienta CVS.
Una herramienta de control de versiones, como CVS, es un sistema de mantenimiento de código fuente (grupos de archivos en general) extraordinariamente útil para grupos de desarrolladores que trabajan cooperativamente usando alguna clase de red. CVS permite a un grupo de desarrolladores trabajar y modificar concurrentemente ficheros organizados en proyectos. Esto significa que dos o más personas pueden modificar un mismo fichero sin que se pierdan los trabajos de ninguna. Además, las operaciones más habituales son muy sencillas de usar.

CVS utiliza una arquitectura cliente-servidor: un servidor guarda la versión actual del proyecto y su historia, y los clientes conectan al servidor para sacar una copia completa del proyecto, trabajar en esa copia y entonces ingresar sus cambios. Típicamente, cliente y servidor conectan utilizando Internet, pero cliente y servidor pueden estar en la misma máquina. El servidor normalmente utiliza un sistema operativo similar a Unix, mientras que los clientes CVS pueden funcionar en cualquier de los sistemas operativos más difundidos.
Los clientes pueden también comparar diferentes versiones de ficheros, solicitar una historia completa de los cambios, o sacar una "foto" histórica del proyecto tal como se encontraba en una fecha determinada o en un número de revisión determinado. Muchos proyectos de código abierto permiten el "acceso de lectura anónimo", significando que los clientes pueden sacar y comparar versiones sin necesidad de teclear una contraseña; solamente el ingreso de cambios requiere una contraseña en estos escenarios.

Los clientes también pueden utilizar el comando de actualización con el fin de tener sus copias al día con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.

El sistema de control de versiones está formado por un conjunto de componentes:
Repositorio: Es el lugar de almacenamiento de los datos de los proyectos. Suele ser un directorio en algún ordenador.

Módulo: En un directorio específico del repositorio. Puede identificar una parte del proyecto o ser el proyecto por completo.
Revisión: Es cada una de las versiones parciales o cambios en los archivos o repositorio completo. La evolución del sistema se mide en revisiones. Cada cambio se considera incremental.
Etiqueta: Información textual que se añada a un conjunto de archivos o a un módulo completo para indicar alguna información importante.
Rama: Revisiones paralelas de un módulo para efectuar cambios sin tocar la evolución principal. Se suele emplear para pruebas o para mantener los cambios en versiones antiguas.

Las órdenes que se pueden ejecutar son:
checkout: obtiene una copia del trabajo para poder trabajar con ella.
Update: actualiza la copia con cambios recientes en el repositorio.
Commit: almacena la copia modificada en el repositorio.
Abort: abandona los cambios en la copia de trabajo.

Repositorio: El repositorio es la parte fundamental de un sistema de control de versiones. Almacena toda la información y datos de un proyecto.
El repositorio es un almacén general de versiones. En la mayoría de las herramientas de control de versiones, suele ser un directorio.
El repositorio centraliza todos los componentes de un mismo sistema, incluyendo las distintas versiones de cada componente. Con el repositorio, se va a conseguir un ahorro de espacio de almacenamiento, ya que estamos evitando guardar por duplicado, los elementos que son comunes a varias versiones.
El repositorio nos va a facilitar el almacenaje de la información de la evolución del sistema, ya que, aparte de los datos en sí mismo, también almacena información sobre las versiones, temporización, etc.

El entorno de desarrollo integrado **NetBeans** usa como sistema de control de versiones CVS. Este sistema, tiene un componente principal, que es el repositorio. En el repositorio se deberán almacenar todos los ficheros de los proyectos, que puedan ser accedidos de forma simultánea por varios desarrolladores.
Cuando usamos un sistema de control de versiones, trabajamos de forma local, sincronizándonos con el repositorio, haciendo los cambios en nuestra copia local, realizando el cambio, se acomete el cambio en el repositorio. Para realizar la sincronización, en el entorno NetBeans, lo realizamos de varias formas:
Abriendo un proyecto CVS en el IDE (entorno de desarrollo integrado).
Comprobando los archivos de un repositorio.
Importando los archivos hacia un repositorio.

Si tenemos un proyecto CVS versionado, con el que hemos trabajado, podemos abrirlo en el IDE y podremos acceder a las características de versionado. El IDE escanea nuestros proyectos abiertos y si contienen directorios CVS, el estado del archivo y la ayuda-contextual se activan automáticamente para los proyectos de versiones CVS.

1.2. Herramientas de control de versiones.

Durante el proceso de desarrollo de software, donde todo un equipo de programadores está colaborando en el desarrollo de un proyecto software, los cambios son continuos. Es por ello necesario que existan en todos los lenguajes de programación y en todos los entornos de programación, herramientas que gestionen el control de cambios.
Si nos centramos en Java, actualmente destacan dos herramientas de control de cambios: CVS y Subversion. CVS es una herramienta de código abierto ampliamente utilizada en numerosas organizaciones. Subversion es el sucesor natural de CVS, está rápidamente integrándose en los nuevos proyectos Java, gracias a sus características que lo hacen adaptarse mejor a las modernas prácticas de programación Java. Estas dos herramienta de control de versiones, se integran perfectamente en los entornos de desarrollado para Java, como NetBeans y Eclipse.

Otras herramientas de amplia difusión son:
SourceSafe: Es una herramienta que forma parte del entorno de desarrollo Microsoft Visual Studio.
Visual Studio Team Foundation Server: Es el sustituto de Source Safe. Es un productor que ofrece control de código fuente, recolección de datos, informes y seguimiento de proyectos, y está destinado a proyectos de colaboración de desarrollo de software.
Darcs: Es un sistema de gestión de versiones distribuido. Algunas de sus características son: la posibilidad de hacer commits locales (sin conexión), cada repositorio es una rama en sí misma, independencia de un servidor central, posibilidad de renombrar ficheros, varios métodos de acceso como local, ssh, http y ftp, etc.
Git: Esta herramienta de control de versiones, diseñada por Linus Torvalds,
Mercurial: Esta herramienta funciona en Linux, Windows y Mac OS X, Es un programa de línea de comandos. Es una herramienta que permite que el desarrollo se haga distribuido, gestionando de forma robusta archivos de texto y binarios. Tiene capacidades avanzadas de ramificación e integración. Es una herramienta que incluye una interfaz web para su configuración y uso.

1.3. Planificación de la gestión de configuraciones.

La Gestión de Configuraciones del software (GCS) es un conjunto de actividades desarrolladas para gestionar los cambios a lo largo del ciclo de vida.
La planificación de la Gestión de Configuraciones del software está regulado en el estándar IEEE 828.
Cuando se habla de la gestión de configuraciones, se está haciendo referencia a la evolución de todo un conjunto de elementos. Una configuración es una combinación de versiones particulares de los componentes que forman un sistema consistente. Desde el punto de vista de la evolución en el tiempo, es el conjunto de las versiones de los objetos componentes en un instante dado.
Una configuración puede cambiar porque se añaden, eliminan o se modifican elementos. También puede cambiar, debido a la reorganización de los componentes, sin que estos cambien.
Como consecuencia de lo expuesto, es necesario disponer de un método, que nos permita designar las diferentes configuraciones de manera sistemática y planificada. De esta forma se facilita el desarrollo de software de manera evolutiva, mediante cambios sucesivos aplicados a partir de una configuración inicial hasta llegar a una versión final aceptable del producto.
La Gestión de Configuraciones de Software se va a componer de cuatro tareas básicas:

1. **Identificación**. Se trata de establecer estándares de documentación y un esquema de identificación de documentos.
2. **Control de cambios**. Consiste en la evaluación y registro de todos los cambios que se hagan de la configuración software.
3. **Auditorías de configuraciones**. Sirven, junto con las revisiones técnicas formales para garantizar que el cambio se ha implementado correctamente.
4. **Generación de informes**. El sistema software está compuesto por un conjunto de elementos, que evolucionan de manera individual, por consiguiente, se debe garantizar la consistencia del conjunto del sistema.

La planificación de la Gestión de Configuraciones de Software va a comprender diferentes actividades:

1. Introducción (propósito, alcance, terminología).
2. Gestión de GCS (organización, responsabilidades, autoridades, políticas aplicables, directivas y procedimientos).
3. Actividades GCS (identificación de la configuración, control de configuración, etc.).
4. Agenda GCS (coordinación con otras actividades del proyecto).
5. Recursos GCS (herramientas, recursos físicos y humanos).
6. Mantenimiento de GCS.

1.4. Gestión del cambio.

Las herramientas de control de versiones no garantizan un desarrollo razonable, si cualquier componente del equipo de desarrollo de una aplicación puede realizar cambios e integrarlos en el repositorio sin ningún tipo de control. Para garantizar que siempre disponemos de una línea base para continuar el desarrollo, es necesario aplicar controles al desarrollo e integración de los cambios. El control de cambios es un mecanismo que sirve para la evaluación y aprobación de los cambios hechos a los elementos de configuración del software.

Pueden establecerse distintos tipos de control:

1. **Control individual**, antes de aprobarse un nuevo elemento. Cuando un elemento de la configuración está bajo control individual, el programador responsable cambia la documentación cuando se requiere. El cambio se puede registrar de manera informal, pero no genera ningún documento formal.
2. **Control de gestión u organizado**, conduce a la aprobación de un nuevo elemento. Implica un procedimiento de revisión y aprobación para cada cambio propuesto en la configuración. Como en el control individual, el control a nivel de proyecto ocurre durante el proceso de desarrollo, pero es usado después de que haya sido aprobado un elemento de la configuración software. El cambio es registrado formalmente y es visible para la gestión.
3. **Control formal**, se realiza durante el mantenimiento. Ocurre durante la fase de mantenimiento del ciclo de vida software. El impacto de cada tarea de mantenimiento se evalúa por un Comité de Control del cuál aprueba las modificaciones de la configuración software.

1.5. Gestión de versiones y entregas.

Las versiones hacen referencia a la evolución de un único elemento, dentro de un sistema software.
La evolución puede representarse en forma de grafo, donde los nodos son las versiones y los arcos corresponden a la creación de una versión a partir de otra ya existente.

Grafo de evolución simple: Las revisiones sucesivas de un componente dan lugar a una simple secuencia lineal. Esta evolución no presenta problemas en la organización del repositorio y las versiones se designan mediante números correlativos.
Variantes: En este caso, existen varias versiones del componente. El grafo ya no es una secuencia lineal, si no que adopta la forma de un árbol. La numeración de las versiones requerirá dos niveles. El primer número designa la variante (línea de evolución) y el segundo la versión particular (revisión) a lo largo de dicha variante.

La terminología que se usa para referirse a los elementos del grafo son:
Tronco (trunk): Es la variante principal.
Cabeza (head): Es la última versión del tronco.
Ramas (branches): Son las variantes secundarias.
Delta: Es el cambio de una revisión respecto a la anterior.

Propagación de cambios: Cuando se tienen variantes que se desarrollan en paralelo, suele ser necesario aplicar un mismo cambio a varias variantes.
Fusión de variantes: En determinados momentos puede dejar de ser necesario mantener una rama independiente. En esta caso se puede fundir con otra (MERGE).
Técnicas de almacenamiento: Como en la mayoría de los casos, las distintas versiones tienen en común gran parte de su contenido, se organiza el almacenamiento para que no se desaproveche espacio repitiendo los datos en común de varias versiones.
Deltas directos: Se almacena la primera versión completa, y luego los cambios mínimos necesarios para reconstruir cada nueva versión a partir de la anterior.
Deltas inversos: Se almacena completa la última versión del tronco y los cambios necesarios para reconstruir cada versión anterior a partir de la siguiente. En las ramas se mantiene el uso de los deltas directos.
Marcado selectivo: Se almacena el texto refundido de todas las versiones como una secuencia lineal, marcando cada sección del conjunto con los números de versiones que corresponde.

En cuanto a la **gestión de entregas**, en primer lugar, definimos el concepto de entrega como una instancia de un sistema que se distribuye a los usuarios externos al equipo de desarrollo.
La planificación de la entrega se ocupa de cuándo emitir una versión del sistema como una entrega. La entrega está compuesta por el conjunto de programas ejecutables, los archivos de configuración que definan como se configura la entrega para una instalación particular, los archivos de datos que se necesitan para el funcionamiento del sistema, un programa de instalación para instalar el sistema en el hardware de destino, documentación electrónica y en papel, y, el embalaje y publicidad asociados, diseñados para esta entrega Actualmente los sistemas se entregan en discos ópticos (CD o DVD) o como archivos de instalación descargables desde la red.

1.6. Herramientas CASE para la gestión de configuraciones.

Los procesos de gestión de configuraciones están estandarizados y requieren la aplicación de procedimientos predefinidos, ya que hay que gestionar gran cantidad de datos. Cuando se construye un sistema a partir de versiones de componentes, un error de gestión de configuraciones puede implicar que el software no trabaje correctamente. Por todo ello, las herramientas CASE de apoyo son imprescindibles para la gestión de configuraciones.

Las herramientas se pueden combinar con entornos de trabajo de gestión de configuraciones.
Hay dos tipos de entornos de trabajo de Gestión de Configuraciones:

- **Entornos de trabajo abiertos**: Las herramientas de cada esta de Gestión de Configuraciones son integradas de acuerdo con procedimientos organizacionales estándar.

Nos encontramos con bastantes herramientas de Gestión de Configuraciones comerciales y open-source disponibles para propósitos específicos. La gestión de cambios se puede llevar a cabo con herramientas de seguimiento (bug-tracking) como **Bugzilla**, la gestión de versiones a través de herramientas como **RCS** o **CVS**, y la construcción del sistema con herramientas como **Make** o **Imake**. Estas herramientas son open-source y están disponibles de forma gratuita.

- **Entornos integrados**: Estos entorno ofrecen facilidades integradas para gestión de versiones, construcción del sistema o seguimiento de los cambios. Por ejemplo, está el proceso de control de Cambios Unificado de **Rational**, que se basa en un entorno de Gestión de Configuraciones que incorpora **ClearCase** para la construcción y gestión de versiones del sistema y **ClearQuest** para el seguimiento de los cambios. Los entornos de Gestión de Configuraciones integrado, ofrecen la ventaja de un intercambio de datos sencillos, y el entorno ofrece una base de datos de Gestión de Configuraciones integrada.

1.7. Clientes de control de versiones integrados en el entorno de desarrollo.

Los entornos de desarrollo actuales integran de forma generalizada, clientes de control de versiones. En algunos caso, como puede ser el IDE Microsoft Visual Studio, se utiliza Visual Studio Team Foundation, de tal forma, que se realiza una gestión centralizada de un proyecto desarrollado por un

equipo, incluyendo la gestión de configuraciones de software, la gestión de versiones, y demás aspectos de un desarrollo en equipo.
Los entorno de desarrollo Open-Source, como Eclipse y NetBeans, los integran de manera directa, o bien instalándolos como plug-in (complementos). Los clientes de control de versiones más destacados son:

- CVS
- Subversion
- Mercurial.

2. Documentación.

El proceso de documentación de código es uno de los aspectos más importantes de la labor de un programador.
Documentar el código nos sirve para explicar su funcionamiento, punto por punto, de forma que cualquier persona que lea el comentario, puede entender la finalidad del código.
La labor de documentación es fundamental para la detección de errores y para su mantenimiento posterior, que, en muchos casos, es realizado por personas diferentes a las que intervinieron en su creación. Hay que tener en cuenta que todos los programas tienen errores y todos los programas sufren modificaciones a lo largo de su vida.
La documentación añade explicaciones de la función del código, de las características de un método, etc. Debe tratar de explicar todo lo que no resulta evidente. Su objetivo no es repetir lo que hace el código, sino explicar por qué se hace.
La documentación explicará cual es la finalidad de una clase, de un paquete, qué hace un método, para que sirve una variable, qué se espera del uso de una variable, qué algoritmo se usa, por qué hemos implementado de una manera y de otro, qué se podría mejorar en el futuro, etc.

2.1. Uso de comentarios.

Uno de los elementos básicos para documentar código, es el uso de comentarios. Un comentario es una anotación que se realiza en el código, pero que el compilador va a ignorar, sirve para indicar a los desarrolladores de código diferentes aspectos del código que pueden ser útiles. En principio, los comentarios tienen dos propósitos diferentes:
Explicar el objetivo de las sentencias. De forma que el programador o programadora, sepa en todo momento la función de esa sentencia, tanto si lo diseñaron como si son otros los que quieren entenderlo o modificarlo.
Explicar qué realiza un método, o clase, no cómo lo realiza. En este caso, se trata de explicar los valores que va a devolver un método, pero no se trata de explicar cómo se ha diseñado.

En el caso del lenguaje Java, C# y C, los comentarios, se implementan de forma similar. Cuando se trata de explicar la función de una sentencia, se usan los caracteres **//** seguidos del comentario, o con los caracteres **/*** y ***/**, situando el comentario entre ellos: ***/* comentario */***
Otro tipo de comentarios que se utilizan en Java, son los que se utilizan para explicar qué hace un código, se denominan comentarios JavaDoc y se escriben empezando por **/**** y terminando con ***/** , estos comentarios pueden ocupar varias líneas. Este tipo de comentarios tienen que seguir una estructura prefijada.
Los comentarios son obligatorios con JavaDoc, y se deben incorporar al principio de cada clase, al principio de cada método y al principio de cada variable de clase. No es

obligatorio, pero en muchas situaciones es conveniente, poner los comentarios al principio de un fragmento de código que no resulta lo suficientemente claro, a la largo de bucles, o si hay alguna línea de código que no resulta evidente y pueda llevarnos a confusión.
Hay que tener en cuenta, que, si el código es modificado, también se deberán modificar los comentarios.

2.2. Alternativas.

En la actualidad, el desarrollo rápido de aplicaciones, en muchos casos, va en detrimento de una buena documentación del código. Si el código no está documentado, puede resultar bastante difícil de entender, y por tanto de solucionar errores y de mantenerlo.
La primera alternativa que surge para documentar código, son los comentarios. Con los comentarios, documentamos la funcionalidad de una línea de código, de un método o el comportamiento de una determinada clase.
Existen diferentes herramientas que permiten automatizar, completar y enriquecer nuestra documentación.
Podemos citar JavaDoc, SchemeSpy y Doxygen, que producen una documentación actualizada, precisa y utilizable en línea, incluyendo, además, con SchemeSpy y Doxygen, modelos de bases de datos gráficos y diagramas.
Insertando comentario en el código más difícil de entender, y utilizando la documentación generada por alguna de las herramientas citadas anteriormente, se genera la suficiente información para ayudar a cualquier nuevo programador o programadora.

2.3. Documentación de clases.

Las clases que se implementan en una aplicación deben de incluir comentarios. Al utilizar un entorno de programación para la implementación de la clase, debemos seguir una serie de pautas, muchas de las cuales las realiza el IDE de forma trasparente, en el diseño y documentación del código. Cuando se implementa una clase, se deben incluir comentarios. En el lenguaje Java, los criterios de documentación de clases, son los establecidos por JavaDoc.
Los comentarios de una clase deben comenzar con **/**** y terminar con ***/** . Entre la información que debe incluir un comentario de clase debe incluirse, al menos las etiquetas **@autor** y **@version**, donde **@autor** identifica el nombre del autor o autora de la clase y **@version**, la identificación de la versión y fecha.
Con el uso de los entornos de desarrollo, las etiquetas se añaden de forma automática, estableciendo el **@autor** y la **@version** de la clase de forma transparente al programador-programadora. También se suele añadir la etiqueta **@see**, que se utiliza para referenciar a otras clases y métodos.

Dentro de la la clase, también se documentan los constructores y los métodos. Al menos se indican las etiquetas:
@param: seguido del nombre, se usa para indicar cada uno de los parámetros que tienen el constructor o método.
@return: si el método no es void, se indica lo que devuelve.
@exception: se indica el nombre de la excepción, especificando cuales pueden lanzarse.
@throws: se indica el nombre de la excepción, especificando las excepciones que pueden lanzarse.

Los campos de una clase, también pueden incluir comentarios, aunque no existen etiquetas obligatorias en JavaDoc.

2.4. Herramientas.

Los entornos de programación que implementa Java, como Eclipse o Netbeans, incluyen una herramienta que va a generar páginas HTML de documentación a partir de los comentarios incluidos en el código fuente. La herramienta ya se ha indicado en los puntos anteriores, y es JavaDoc. Para que JavaDoc pueda generar las páginas HTML es necesario seguir una serie de normas de documentación en el código fuente, estas son:
Los comentarios JavaDoc deben empezar por /** y terminar por */.
Los comentarios pueden ser a nivel de clase, a nivel de variable y a nivel de método.
La documentación se genera para métodos public y protected.
Se puede usar tag para documentar diferentes aspectos determinados del código, como parámetros Los tags más habituales son los siguientes:

Tipo de tag	Formato	Descripción
Todos.	@see.	Permite crear una referencia a la documentación de otra clase o método.
Clases.	@version.	Comentario con datos indicativos del número de versión.
Clases.	@author.	Nombre del autor.
Clases.	@since.	Fecha desde la que está presente la clase.
Métodos.	@param.	Parámetros que recibe el método.
Métodos.	@return.	Significado del dato devuelto por el método
Métodos.	@throws.	Comentario sobre las excepciones que lanza.
Métodos.	@deprecated.	Indicación de que el método es obsoleto.

2.4.1. Documentación. Uso de comentarios. Alternativas.

La documentación es el texto escrito que acompaña a los proyectos. Es un requisito

importante en un proyecto comercial, ya que el cliente querrá que se documente las distintas fases del proyecto.
Podemos distinguir entre los siguientes tipos de documentación:

- **Documentación de las especificaciones:** sirve para comprobar que tanto las ideas del desarrollador como las del cliente son las mismas, ya que sino el proyecto no será aceptable. Según la norma IEEE 830, que recoge varias recomendaciones para la documentación de software, esta documentación deberá contener:
  - ***Introducción****: s*e explican los fines y objetivos del software.
  - ***Descripción de la información****: d*escripción detallada, incluyendo hardware y software.
  - ***Descripción funcional****: d*etalla cada función del sistema.
  - ***Descripción del comportamiento****: e*xplica cómo se comporta el software ante sucesos externos e internos.
  - ***Criterios de validación****: d*ocumento sobre el límite de rendimientos, los tipos de pruebas, la respuesta esperada del software y las consideraciones especiales.
- **Documentación del diseño:** en este documento se describe toda la estructura interna del programa, formas de implementación, contenido de clases, métodos, objetos, etc.
- **– Documentación del código fuente:** durante el desarrollo del proyecto se debe ir comentando en el código fuente cada parte, para tener una mayor claridad de lo que se quiere conseguir en cada sección.
- **Documentación de usuario final:** documentación que se entrega al cliente en la que se describe cómo usar las aplicaciones del proyecto.

2.4.2. Documentación del código fuente

La documentación del código del programa también es fundamental para que todo el equipo pueda realizar funciones de actualización y reparación de errores de manera mucho más sencilla.
Esta debe describir lo que se está haciendo y por qué. Hay 2 reglas que no se deben olvidar:

- Todos los programas poseen errores y es cuestión de tiempo que se detecten.
- Todos los programas sufren modificaciones a lo largo de su vida.

Al realizar las modificaciones es necesario que el código esté bien documentado para que otro programador ajeno localice los cambios que quiere realizar.
Al documentarlo, habrá que explicar lo que realiza una clase o un método y por qué y para qué lo hace.
Para documentar proyectos existen muchas herramientas como pueden ser PHPDoc, phpDocumentor, Javdoc o JSDoc, el javadoc para JavaScript. Nosotros usaremos Javadoc.

2.4.3. Uso de Javadoc en Eclipse

Javadoc es una herramienta de Java que sirve para extraer y generar documentación básica para el programador a partir del código fuente en formato HTML. Tendremos que escribir los comentarios siguiendo las recomendaciones de Javadoc, y el código y la documentación se encontrarán dentro del mismo fichero.
Los tipos de comentario para que genere la documentación son:
− **Comentarios en línea:** comienzan con los caracteres "//" y terminan en la misma línea.
− **Comentarios tipo C:** comienzan con "/*" y terminan con "*/". Pueden contener varias líneas.
− **Comentarios de documentación Javadoc:** se colocan entre delimitadores /**…*/, podrán agrupar varias líneas y cada línea irá precedida por un *.

Deberá colocarse antes de la declaración de una clase, un campo, un método o un constructor. Podrá contener etiquetas HTML y los comentarios están formados por dos partes: una descripción seguida de un bloque de *tags*.

Uso de etiquetas de documentación

Las etiquetas de Javadoc van precedidas por @ y las más utilizadas son:
ETIQUETA DESCRIPCIÓN
@author Autor de la clase. Solo para clases
@version Versión de la clase. Solo para clases
@see Referencia a otra clase
@param Descripción del parámetro. Una etiqueta por cada parámetro
@return Descripción de lo que devuelve. Solo si no es *void*.
@throws Descripción de la excepción que puede propagar. Habrá una etiqueta *throws* por cada tipo de excepción
@since Número de versión de la que existe el método

Generar la documentación

Casi todos los entornos de desarrollo incluyen un botón para poder configurar Javadoc. Para hacerlo desde Eclipse, abrimos el menú *Project* y elegimos el botón *Generate Javadoc*. En la siguiente ventana nos pedirá la siguiente información:
− En *Javadoc command* se indicará dónde se encuentra el fichero ejecutable de Javadoc, el javadoc.exe. Pulsamos en *Configure* para buscarlo dentro de la carpeta del JDK y elegimos la carpeta bin.
− En los cuadros inferiores elegiremos el proyecto y las clases a documentar.
− Elegimos la privacidad de los elementos. Con *Private* se documentarán todos los miembros públicos, privados y protegidos.

− Para finalizar, se indica la carpeta de destino en la que se almacenará el código HTML.

Pulsar en *Next,* en la siguiente ventana poner el título del documento html que se genera, y elegir las opciones para la generación de las páginas HTML. Como mínimo se seleccionará la barra de navegación y el índice.