U6 - Optimización y documentación

Una parte importante en el desarrollo del software además de la funcionabilidad del producto, es su optimización, mantenimiento y evolución.

El programador, aunque siga un plan en su elaboración siempre pueden quedar líneas de código que no estén bien estructuradas o veamos que nos falta un método o atributo.

Es por ello que una vez terminado y siendo ya funcional, nos encontramos con que podemos darle una estructura más clara. Aunque ese proceso se podría hacer de forma manual, en algunos casos puede ser una tarea muy ardua.

SOLUCIÓN: La mayoría de los IDE proporcionan utilidades para ayudarnos: Mediante la refactorización podemos realizar cambios fácilmente y optimizar nuestro código

Otro punto importante clave: Un código bien documentado. Norma: 60 % de nuestro código lo conformen comentarios.

OBJETIVOS:

1. Conocer cuáles son los métodos y las acciones que se pueden realizar en el refactoring y para qué se realizan.
2. Manejar las técnicas de refactorización.
3. Conocer los sistemas de control de versiones.
4. Conocer las herramientas disponibles en los entornos de desarrollo para el control de versiones.
5. Comprender cómo generar documentación para nuestros proyectos.

6.1 REFACTORIZACIÓN.

La refactorización o refactoring tiene como objetivo la restructuración del código fuente, de forma que cambia lo escrito por nuevo código, pero sin cambiar ningún aspecto de su funcionabilidad.

Los entornos de desarrollo disponen de soporte automatizado para la refactorización de nuestras aplicaciones.

OBJETIVO: simplificación de la lógica y la eliminación de niveles innecesarios de complejidad

Refactor permite adaptar todo el código necesario ante un cambio requerido.

OJO: Un error común entre los programadores es pensar que cuanto menor sea el número de líneas, más óptimo será el código. Y, nada más lejos de la realidad, el código para ser óptimo también debe estar bien estructurado, ha de ser fácil de leer y además estar correctamente documentado.

La mayoría de los IDE proporcionan soporte para algunas de las refactorizaciones. Estas herramientas podremos encontrarlas al seleccionar "Refactor" en el menú correspondiente

Entre las técnicas más comunes de refactorización:

- Mejora de nombres y ubicaciones de código o bloques de código.
- Generación de código: Crear construcción, getters and setters.

6.1.1. Herramientas de refactorización:

- Rename
- Move
- Copy
- Safety Delete
- Inline
- Change Method Parameters
- Pull up
- Pull down
- Introduce

Vídeo sobre refactorización: https://vimeo.com/272132634

6. 2. Control de versiones

Un Sistema de Control de Versiones (SCV) nos proporciona diferentes versiones de la codificación de un proyecto. Durante el proceso de desarrollo la aplicación va cambiando y puede que en ocasiones sea necesario volver a un estado anterior.

Con estas versiones el programador puede:

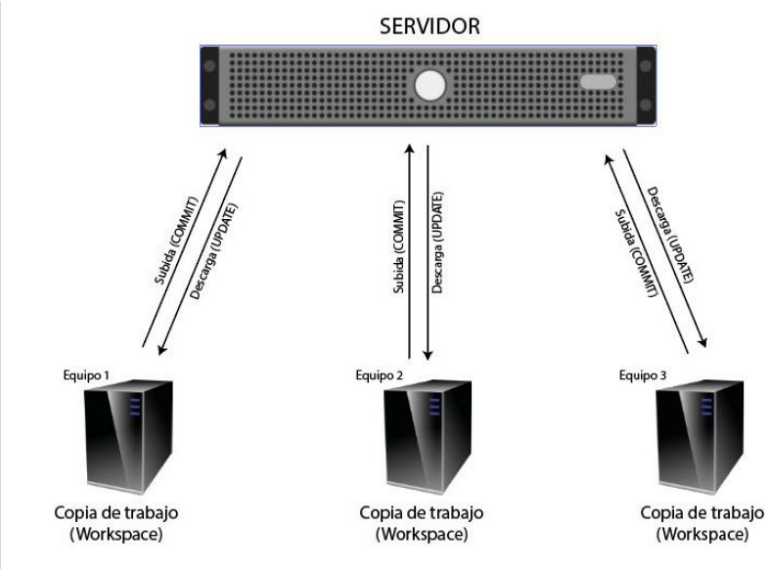
- Restaurar en parte o en su totalidad un desarrollo a una versión anterior.
- Comparar los cambios de la versión actual respecto a otra anterior.
- Acceso remoto: proporcionar acceso remoto de elementos del repositorio a los usuarios con permisos para modificar, leer, mover, borrar, etc.
- Registrar las modificaciones: crear un registro de modificaciones efectuadas sobre la estructura de archivos que permita obtener estados previos.
- Control de cambios: controlar que los usuarios siempre trabajen sobre la última versión del archivo.

6.2.1. Tipos de control de versiones.

Cuando trabajamos con un grupo de personas es común que varias estén trabajando en las mismas partes de un código, por ello tenemos que asegurarnos de que cuando se realicen cambios todos lo estén haciendo en la versión correcta.

1. Sistemas centralizados.

Usuarios de los diferentes equipos, por medio de una orden “update”, obtienen una copia de trabajo de la versión requerida, procedente del repositorio del servidor. Una vez realizado el cambio en la copia de forma local mediante la orden “commit”, se enviarán los cambios realizados en la versión al servidor.



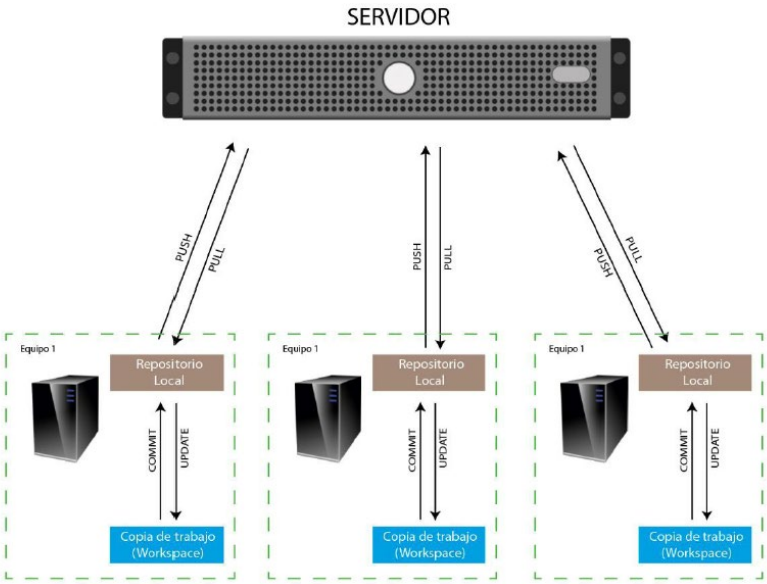
Es posible que dos programadores trabajan sobre el mismo código, entonces será necesario resolver conflictos para quedarse con la versión final tras una resolución de dicho conflicto.

OJO: El repositorio y todo el código se encuentra en el servidor centralizado únicamente al que acceden todos los trabajadores y dejan el resultado de su trabajo, es decir, el cliente en su equipo no tiene un repositorio del todo el código, sino que siempre depende de la disponibilidad del servidor centralizado.

COMANDOS:

- UPDATE: Descargar el código o tarea según la última versión o situación el mismo en el servidor.
- COMMIT: Subir el código al servidor central.

2. Sistemas distribuidos



En los sistemas distribuidos, un usuario accede por primera vez al sistema y, por medio de la orden "pull", descarga el repositorio completo desde el servidor hasta su equipo local

El usuario, por medio de la orden "update" o "pull", obtiene una copia de trabajo procedente del repositorio de su equipo. Una vez realiza los cambios por medio de una orden "commit" guarda los cambios realizados en el repositorio local de su equipo. En este momento, el código todavía no está subido al servidor, sino que únicamente está registrada esta nueva versión de forma local.

Si desea que los cambios de su repositorio local se hagan efectivos en el servidor deberá hacerlo por medio de la orden "push".

COMANDOS:

- PULL. Actualizar o descargar la última versión situada en el servidor central frente a la versión local del cliente.
- COMMIT. Se guarda los cambios dentro ÚNICAMENTE del repositorio local, es decir, dentro del equipo del usuario.
- PUSH. Se trasmite los cambios entre la versión del repositorio en local y el repositorio en el servidor común para todos los programadores.

Ventajas

- Centralizado: El repositorio se gestiona de manera más sencilla.
- Distribuido: Ante una caída del servidor se puede recuperar el repositorio íntegro desde cualquiera de los equipos.

Desventajas

- Centralizado: Ante una caída del servidor central no será posible respaldar los datos. Las copias del repositorio central deben ser más frecuentes al contar con un único repositorio.
- Distribuido: Cuando se propagan los cambios del repositorio de un usuario se obtienen elevados tiempos de actualización y mayor tráfico de datos. Al trabajar con repositorios locales las actualizaciones hacia el resto de usuarios suelen realizarse con menos frecuencia.

6.3. Gestión y administración

Repositorios

Los repositorios son los almacenes donde se guardan los datos, estos son una secuencia de estados de las estructuras de archivos y directorios almacenados a lo largo del tiempo. Inicialmente el repositorio estará vacío, pero según realicemos envíos (commits) se irán añadiendo al repositorio, guardando el estado anterior y el nuevo.

Publicación de cambios «commit» + << push >>

Commit se genera una versión dentro del repositorio local del programador.

Es al hacer el push que aplicamos los cambios dentro de un servidor central propio o el uso de alguna plataforma como Github, Gitlab o Bitbucket.

Despliegue o pull.

Proceso en el que el usuario descarga o actualiza su copia de trabajo o workspace desde el repositorio

Ramas «branching»

Muchas veces en la realización de un proyecto se requiere la realización de versiones en paralelo. Es decir, a partir de una versión común, desarrollarla por dos vías o “ramas” (branches) distintas.

Fusiones "merging"

Las ramas anteriormente mencionadas posteriormente se podrán unificar conformando una única versión que combine los cambios aplicados de forma paralela en más de una rama a este proceso de denomina fusión o “merging”.

Aunque el sistema realice una combinación de forma automática, una fusión siempre deberá ser verificada de forma manual.

Etiquetado («tagging»)

Uno de los motivos de poner una etiqueta o marca a una versión es la de poder identificar un hito o punto importante en el desarrollo del proyecto, como puede ser la finalización de una versión o una entrega del producto al cliente.

6.4. Herramientas para el control de versiones

6.4.1. Sistemas de gestión de versiones

Existen una gran variedad de herramientas que proporcionan un sistema de control de versiones. Cada una incorpora una serie de características que habrán de ser tenidas en cuenta a la hora de seleccionar el sistema.

- Tipo de repositorio que incorpora: central o distribuido.
- Tipo de despliegue: exclusivos o colaborativos.
- Soporte para “commit” atómicos o consolidado, permite realizar la actualización mientras se garantice se cargan automáticamente y se combinan.
- Capacidad de mostrar árboles (branch) de otros sistemas.
- Protocolos de red soportados (HTTP, SSH.FTP, SMTP, POP3).
- Soporte de etiquetas.
- Posibilidad de firmar versiones.
- Soporte con sistemas operativos.
- Integración con herramientas de desarrollo.
- Tipo de licencia.
- Capacidad de almacenamiento del repositorio.

6.4.2. Introducción a GIT

Características:

- Sistema distribuido. INCREMENTAL.
- Permite el uso de ramas o "branches", lo que hace que podamos trabajar en diferentes versiones del código sin tocar el principal
- Con GIT podemos trabajar directamente desde la consola de comandos, aunque también existen herramientas gráficas como Egit en Eclipse.

Ventajas:

- GIT es un software libre bajo la licencia de código abierto GPL. Gratuito.
- GIT no es centralizado, permite usar únicamente el repositorio local.
- Al ser un sistema descentralizado cada cliente es una copia del servidor. Minimiza el flujo de actualización contra el servidor y minimiza la posibilidad de pérdidas de información.
- GIT utiliza una criptográfica (SHA1) para nombrar e identificar objetos, por lo que aporta seguridad
- La gestión de ramas en un servidor GIT es muy rápida y sencilla, al ser incremental no copia todo, sino solo los cambios.

COMPONENTES:

- Repositorio local.
- Workspace o espacio de trabajo.
- Staging Index. Área donde se controla los cambios se indica que ficheros quieres unir al commit y cuales ignorar, permite añadir una etiqueta al commit.
- Repositorio remoto.
- Commits.
- Branches.
- Tags.

COMANDOS PRINCIPALES:

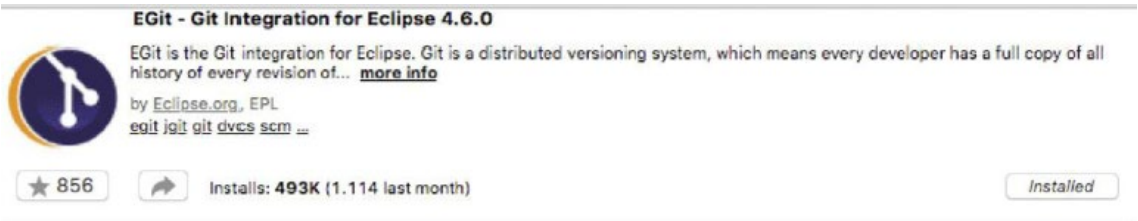
- Add. Añadir ficheros o código al conjunto de datos del respositorio.
- Commit. Guardar una nueva versión del código dentro del respositorio local.
- Push. Trasladar el commit local hacia el servidor.
- Pull. Extraer una copia del servidor, actualizando el repositorio local con información remota, debido a cambios no efectuados por el usuario.
- Chechkout. Copia del repositorio.
- HEAD: Ultimo código realizado.
- Diff HEAD: Muestra los cambios entre la versión nueva o a registrar ya versión ya registrada previamente en el respositorio. Indica los cambios.
- Diff. Permite comparar versiones.

3.4.3. Uso de GIT por consola

- git init: Iniciar repositorio.
- git config --global user.mail "mail"
- git config --gobal user.name "nombre"
- git add: Indicamos que ficheros añadimos al repositorio. Todos: git add.
- git status: Vemos si hay algún fichero pendiente de actualizar en el repositorio local.
- git commit –m "título de la versión"
- git log: Para ver los "commit" que hemos realizado, tenemos disponible el comando "git log":
- git remote add origin https://github.com/IMF-ManuelVazquez/ED_prueba.git
  git push -u origin master
- git pull : Actualizamos nuestro repositorio local con la versión del repositorio en el servidor.

3.4.4. Integrar GIT en Eclipse.

Utilizaremos egit. Los pasos de cómo integrarlo en Eclipse serán explicados en clase al igual del modo de uso del mismo.



3.4.5. Uso de framework de terceros para la gestión de GIT.

Consiste en un software de terceros que permite una gestión de GIT muy sencilla y visual. Existen diversas aplicaciones entre las que destacamos:

- Gitkraken: https://www.gitkraken.com/
- SmartGit: https://www.syntevo.com/smartgit/

3.5. Generar la documentación

Es fundamental documentar correctamente las aplicaciones de software, ya que facilitan la comprensión de las tareas llevadas a cabo por las funciones de modo que todo el equipo comprenda los códigos y elementos utilizados, el mantenimiento y su reutilización.
Podemos realizar comentarios de una única línea o de varias a la vez.

LENGUAJE	DE UNA SOLA LÍNEA	DE VARIAS LÍNEAS
Visual Basic	Comilla al principio '	Comilla al principio '
C, C++, Java, C#	Inicio de comentario con //	Inicio con /* y al final con */
PHP (también los mismos de C)	Inicio de comentario con #	Inicio de comentario con #
Phyton	Inicio de comentario con #	Inicio y fin con tres comillas """
Ruby	Inicio de comentario con #	Inicio con =begin y fin con =end
HTML, XML	Principio con <!- y fin con --> Los lenguajes de script que llevan dentro etiquetas HTML, utilizarán además de este tipo los suyos propios	Principio con <!- y fin con --> Los lenguajes de script que llevan dentro etiquetas HTML, utilizarán además de este tipo los suyos propios
JSP	Principio con <%-- y fin con --%>	Principio con <%-- y fin con --%>

3.5.1. Generador de documentación. Javadoc.

Los lenguajes de programación habitualmente ofrecen herramientas de generación de documentación de forma automática. Java tiene **javadoc**, el cual rastrea los ficheros .java buscando unos comentarios especiales que permiten generar páginas HTML con documentación acerca de estos.

COMANDOS:

- ETIQUETA DE INTERFAZ:
  @author texto: autor de la clase / interfaz.
  @version texto: versión de la clase / interfaz.
  @since texto: indica desde que versión existe.
  @see nombreDeClase: crea un hiperenlace.

- ETIQUETA DE ATRIBUTO: Se deben poner antes de la declaración del atributo:
  @see nombreDeClase: crea un hiperenlace.

- ETIQUETA DEL CONSTRUCTOR O MÉTODOS. Se colocan antes de la declaración de un constructor o un método:
  @param nombreParametro descripcionParametro.
  @exception nombreClase descripcion.
  @return texto.
  @see nombreDeClase.
  @deprecated. Indica que el método es antiguo y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores.

3.5.2. Generador de documentación para otros lenguajes.

Doxygen es un generador de documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft), VHDL y en cierta medida para PHP, C# y D.
Multiplataforma: Linux, Windows y Mac OS X
Formatos de salida: http://www.doxygen.nl/manual/output.html