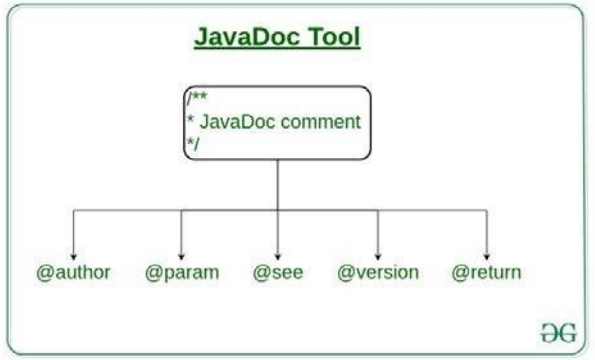
EFA
MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

ENTORNOS DE DESARROLLO

DANIEL GONZÁLEZ-CALERO JIMÉNEZ

UT5 – OPTIMIZACIÓN Y DOCUMENTACIÓN



EFA MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

ENTORNOS DE DESARROLLO

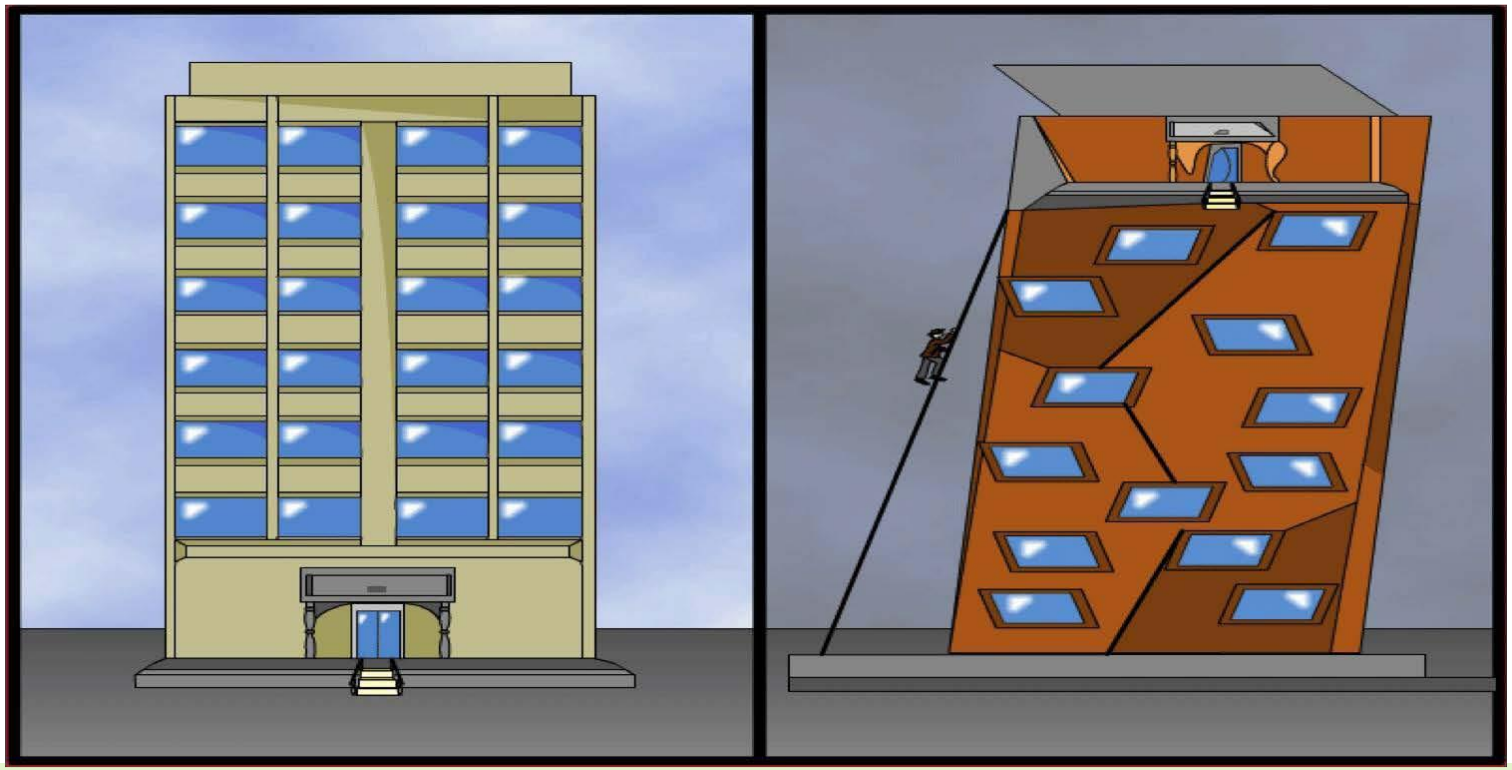
INDICE

UT5 – OPTIMIZACIÓN Y DOCUMENTACIÓN

REFACTORIZACIÓN

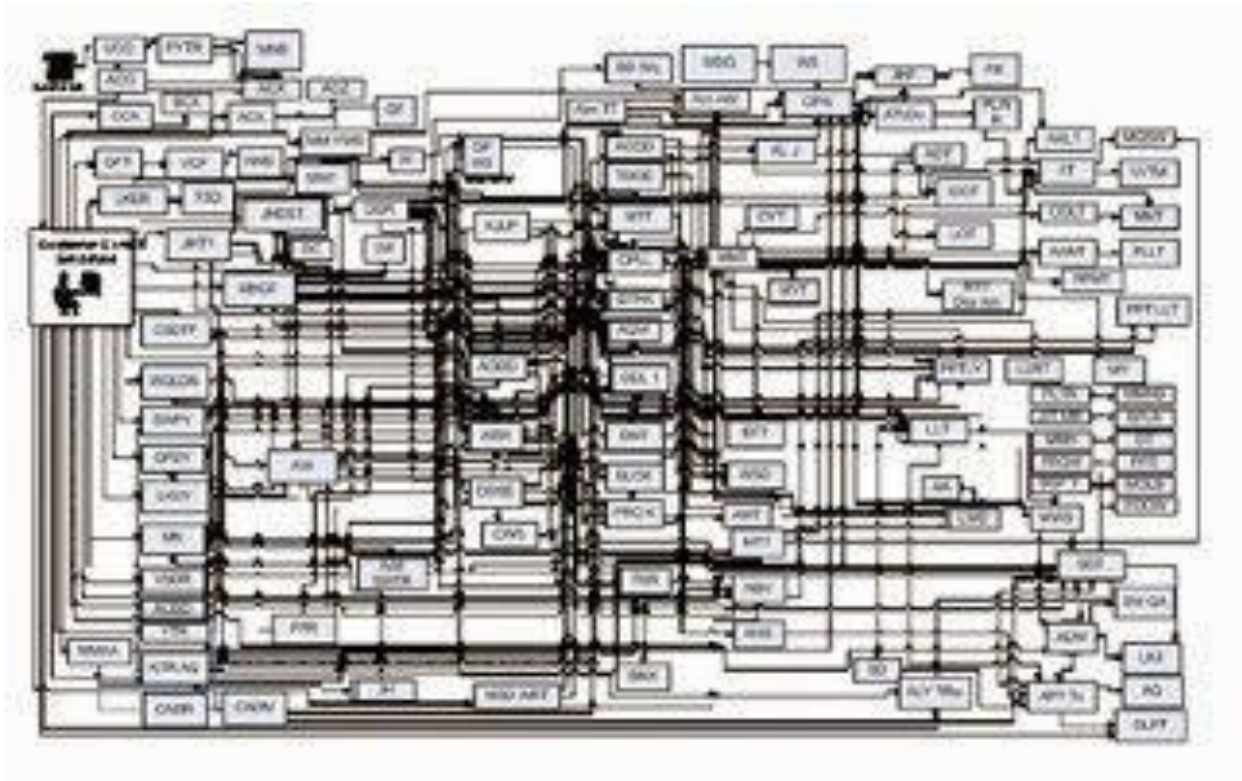
Problemática

- ✓ Si en lugar de programadores fuéramos constructores, ¿cómo serían nuestros edificios?



Problemática

- ¿Qué malas prácticas de codificación tiene un edificio como el de la derecha?
  - Código Spaghetti
  - Diseño roto
  - Código muy grande poco modularizado
  - Falta de documentación



Problemática

- ✓ Debido a las limitaciones de hardware que existían en el pasado, la prioridad era tener un código rápido, pequeño (ocupa poca memoria), optimizado, utilizando un algoritmo eficaz, etc.
- ✓ En la actualidad, debido a las mejoras en el hardware y la complejidad del software, se están generando muchas aplicaciones mal diseñadas que son ineficientes y difíciles de mantener
- ✓ Aproximadamente el 80% del trabajo en desarrollo de software es el mantenimiento y evolución a sistemas existentes

Objetivos a conseguir

- ✓ Desarrollar un software simple en su diseño y codificación
  - ✓ **KISS**
- ✓ Realizar software minimalista sin pérdida de funcionalidad
- ✓ Reestructurar todo el proceso de desarrollo de software (datos, diseño externo, documentación, etc.), no solo el código fuente
- ✓ Crear un proceso de mejora continua:
  - ✓ Las mejoras se conseguirán mejorando el software iteración por iteración
  - ✓ La metodologías ágiles promueven las técnicas de mejora continua

Refactorizar - Definiciones formales

- ✓ *"Mejorar el diseño de código existente"* **Martin Fowler**
- ✓ *"Modificar el comportamiento interno (generalmente código fuente) sin modificar su comportamiento externo (apariencia, funcionalidad)"* **Roger Pressman**
- ✓ *"Realiza una transformación al software preservando su comportamiento, modificando su estructura interna para mejorarlo"* *William Opdyke*

Refactorizar

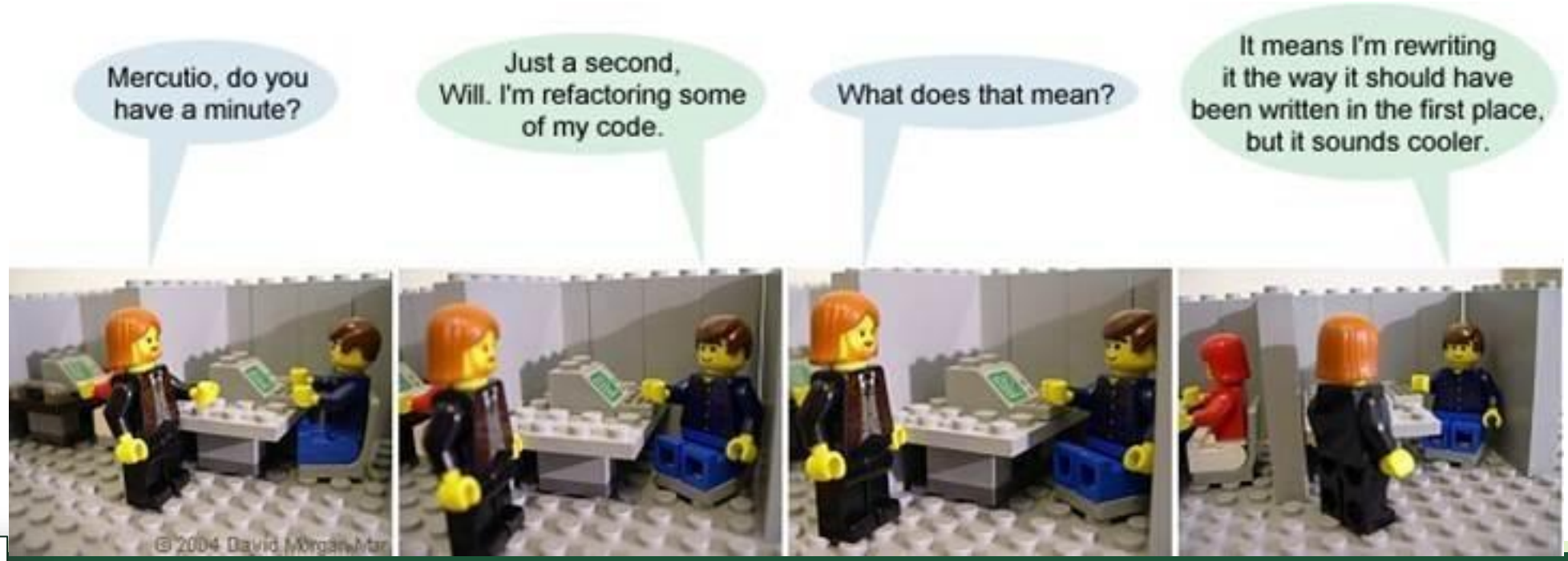
- ✓ La refactorización es llamada informalmente *limpieza de código*
- ✓ Refactorizando se consigue un código fuente:
  - ✓ Más sencillo de comprender
  - ✓ Más compacto
  - ✓ Más limpio
  - ✓ Más fácil de modificar
  - ✓ Más eficiente
- ✓ Refactorizar **NO** cambia la funcionalidad del código ni el comportamiento del programa

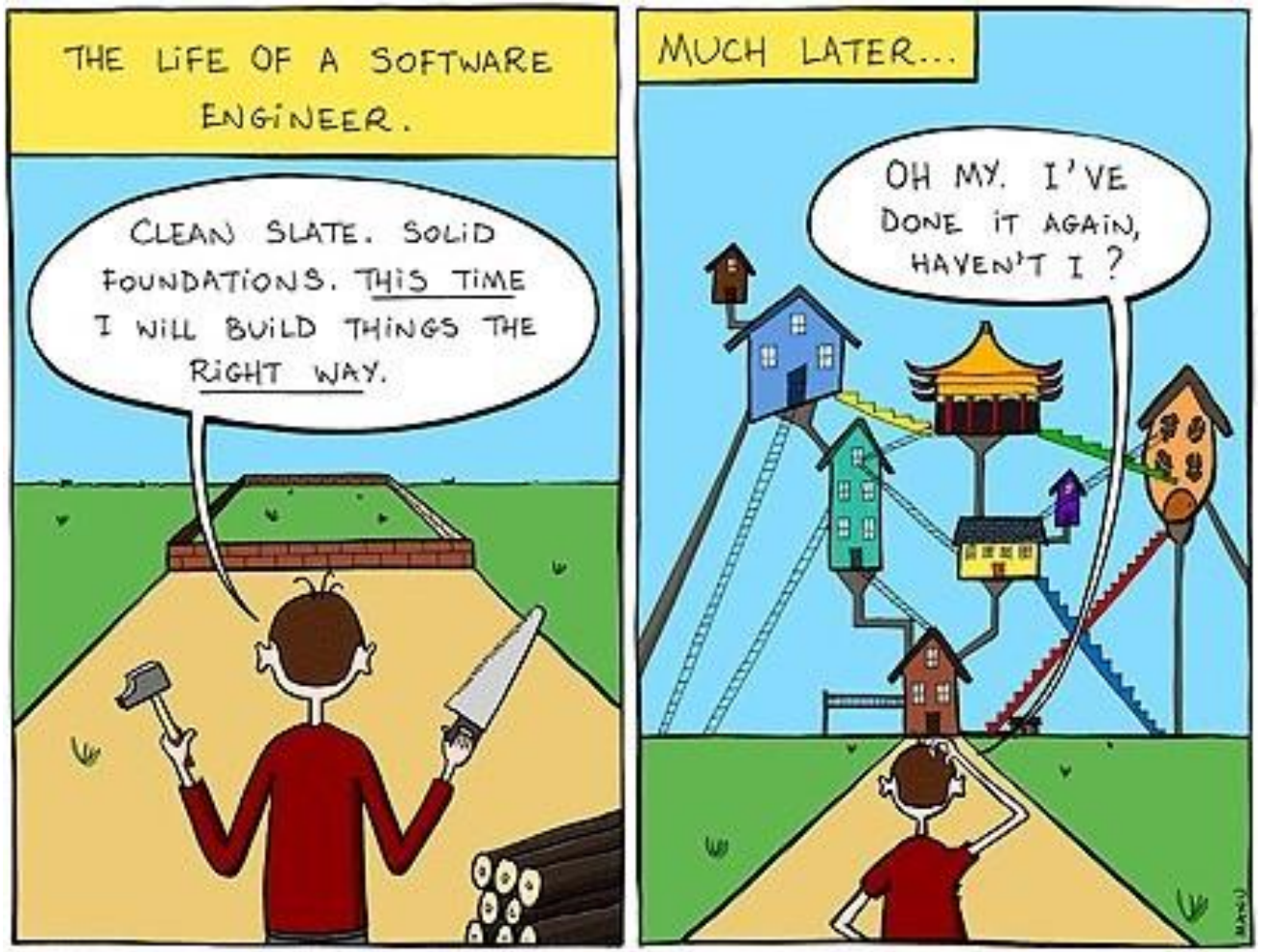
Refactorizar

- ✓ El programa deberá comportarse de la misma forma antes y después de efectuar las refactorizaciones
- ✓ Usar **pruebas unitarias** periódicamente es una buena herramienta para asegurarse que la refactorización no ha modificado ni roto el programa

Refactorizar - ¿Dónde se aplica?

- ✓ En proyectos de grandes proporciones o de larga duración
- ✓ En el código fuente realizado por otra persona
- ✓ En el código fuente que empieza a ser largo, tedioso y difícil de entender




THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?

Refactorizar - ¿Cuándo se aplica?

- ✓ No existe ninguna etapa de desarrollo especifica
- ✓ Siempre que se quiera ya que la funcionalidad no va a sufrir cambio alguno
- ✓ Aunque no exista ninguna etapa especifica para la refactorización, debería ser una tarea recurrente a la hora de codificar o mantener una aplicación
- ✓ El modo más eficiente de tener un código refactorizado es intercalar la creación de código nuevo con la refactorización de código
- ✓ Es común aplicar refactorización cuando el software tiene **bad smell** *(malos olores)*

Malos olores

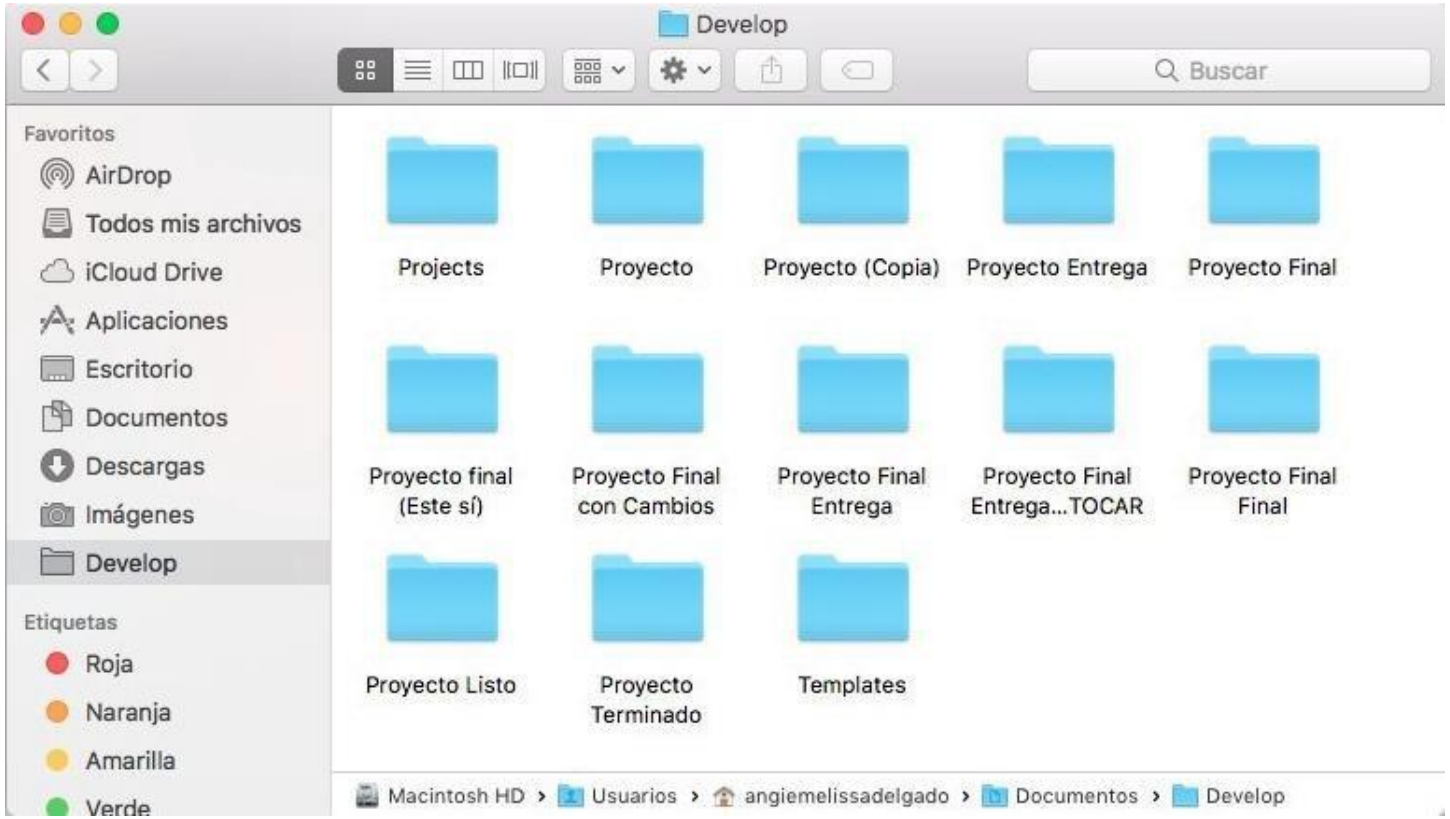
- ✓ Son patrones que se cumplen y evidencian los problemas de diseño de un código fuente:
- ✓ **Métodos largos:**
  - ✓ Los programas con más vida útil son aquellos con métodos cortos, que son más reutilizables y aportan mayor semántica
- ✓ **Clases grandes:**
  - ✓ Clases que hacen demasiado por lo general son muy vulnerables al cambio
- ✓ **Lista de parámetros larga:**
  - ✓ Los métodos con muchos parámetros elevan el acoplamiento, son difíciles de comprender y cambian con frecuencia
- ✓ **Clases sin comportamientos:**
  - ✓ Clases que solo tienen atributos y métodos tipo get y set. Las clases casi siempre deben disponer de algún comportamiento no trivial

Malos olores

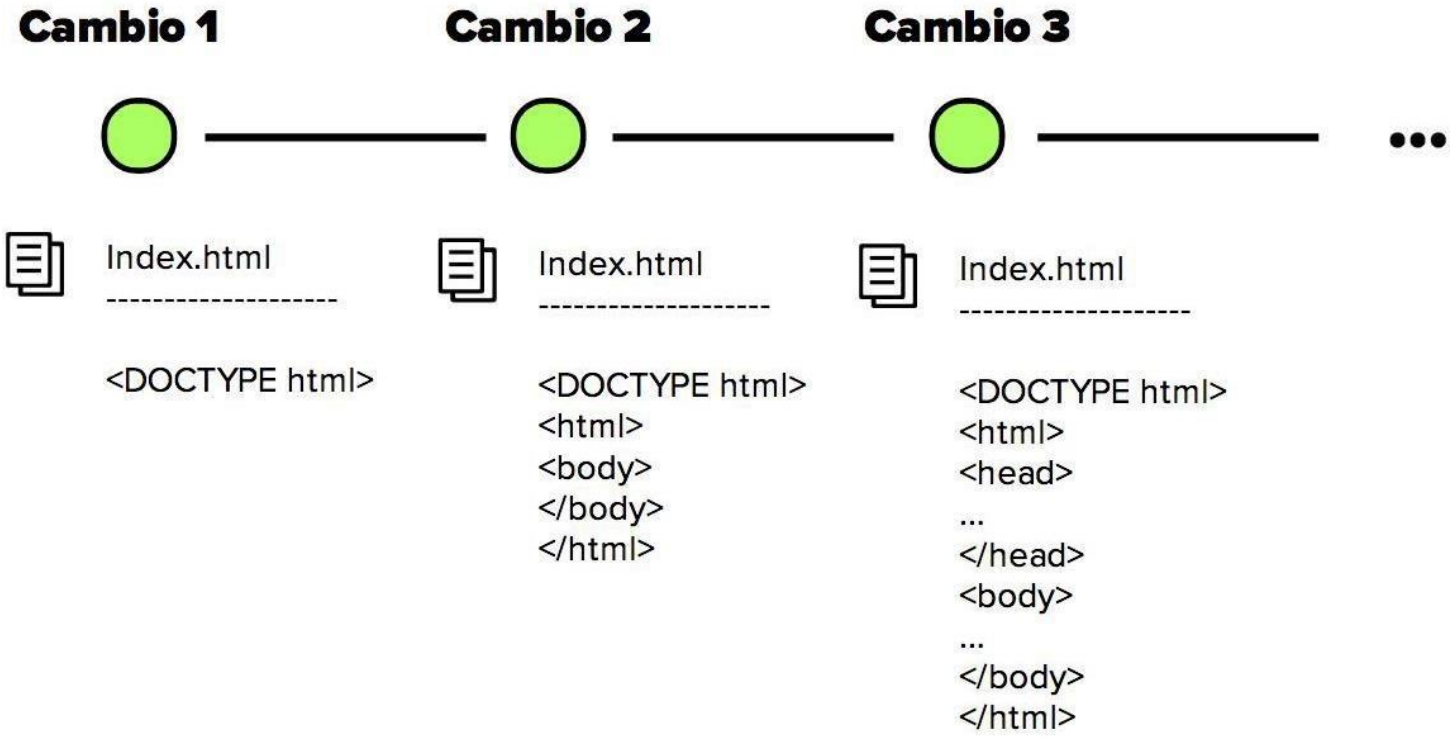
- ✓ **Estructura de agrupación condicional:**
  - ✓ Un *if* con muchos casos, seguramente puede escribirse con una sentencia *switch*
- ✓ **Herencia perdida:**
  - ✓ Clases hijas sin atributos diferentes entre hermanas
- ✓ **Clase perezosa:**
  - ✓ Una clase que no hace nada o casi nada debería eliminarse
- ✓ **Duplicidad de código:**
  - ✓ Código duplicado pudiendo establecer métodos que agrupen esa funcionalidad
- ✓ **Grupos de datos:**
  - ✓ Englobar en tipos de datos complejos tipos de datos simples que pertenezcan a una misma entidad

SISTEMAS DE CONTROL DE VERSIONES

✓ La vida de un desarrollador software antes de conocer los Sistema de Control de Versiones



- ✓ El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo de tal manera que sea posible recuperar versiones especificas más adelante



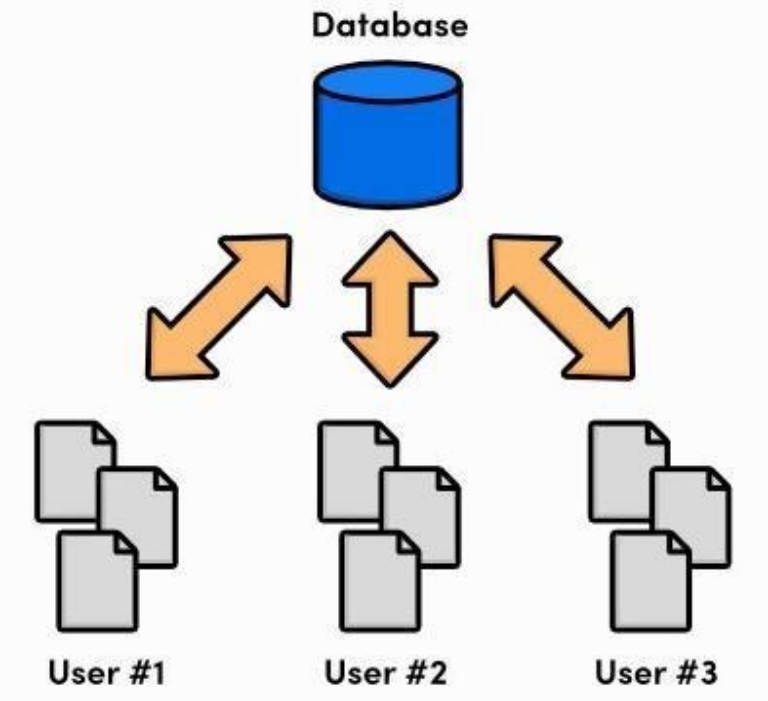
CVS Centralizados

- ✓ El repositorio se almacena en el servidor
- ✓ Es necesaria la conexión al servidor para enviar cambios al repositorio
- ✓ Es necesario menos espacio de almacenamiento en disco
- ✓ Ejemplo: **Subversion - SVN**

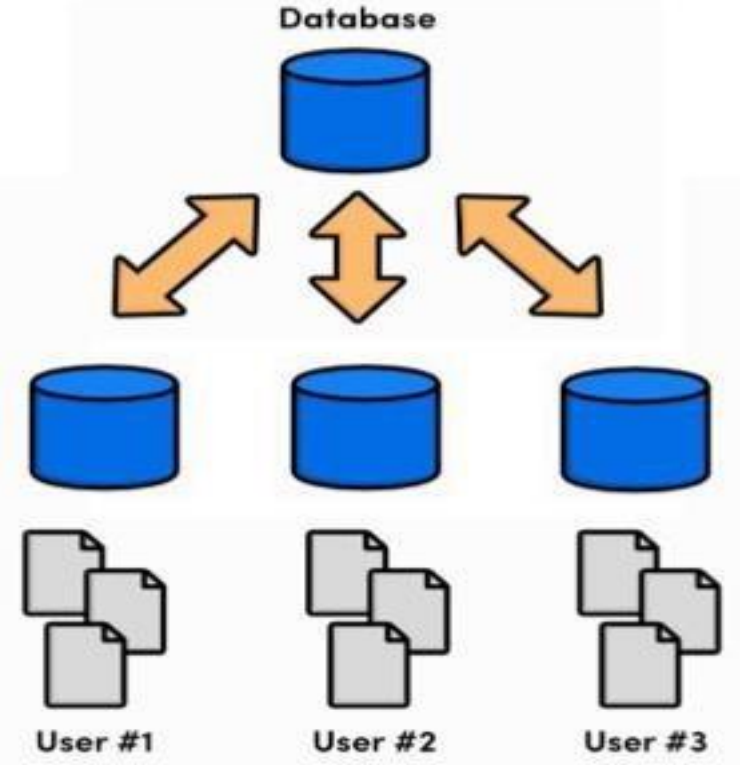
CVS Distribuidos

- ✓ El repositorio se almacena en el servidor y en ordenador de los desarrolladores
- ✓ No es necesaria la conexión al servidor para enviar cambios al repositorio
- ✓ E necesario más espacio de almacenamiento en disco
- ✓ Ejemplo: **Git**

CVS Centralizados



CVS Distribuidos



Subversion

- ✓ **Subversion (SVN)** fue creado por la empresa *CollabNet* en el año 2000
- ✓ Actualmente desarrollado por la Apache Software Foundation
- ✓ Uno de los SCV más utilizados históricamente
- ✓ Sistema de control de versiones centralizado
- ✓ Existe un *repositorio* que dispone de todo el historial de las versiones del software
- ✓ Los desarrolladores deben trabajar en red y conectarse al repositorio central para sincronizar los ficheros del proyecto

Subversion

- ✓ Trabaja con archivos y directorios
- ✓ Las copias, eliminaciones y renombrados de los archivos son versionados
- ✓ La revisión del software se hace por *commit* y no para cada archivo o directorio particular
- ✓ Sencilla creación de *ramas* y *tags*
- ✓ Resolución interactiva de *conflictos*, que permite mezclar el código de los archivos versionados de manera sencilla, tanto desde la línea de comandos como utilizando diversos programas de interfaz gráfica

Subversion

- ✓ Ejemplos de programas con interfaz gráfica para trabajar con Subversion

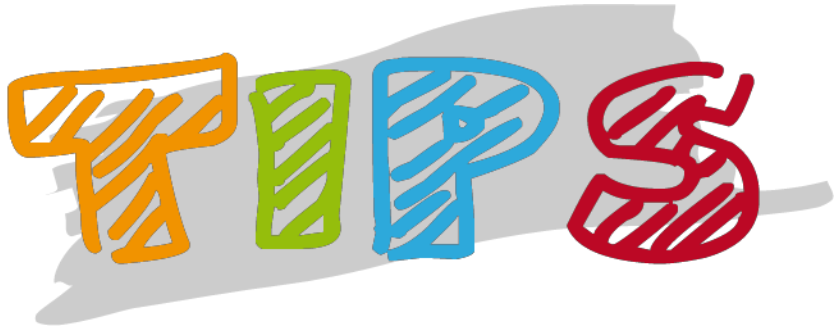
Subclipse

DOCUMENTACIÓN

- ✓ Todos los programas deben estar adecuadamente documentados
- ✓ Documentar un programa es tedioso; pero absolutamente necesario para aumentar su legibilidad
- ✓ Si un código tiene una alta calidad de programación puede estar exento de comentarios, pues está tan bien escrito que se entiende solo con leer el código fuente

Consejos para realizar los comentarios

1. Los comentarios tienen que ser útiles
2. Piensa en un comentario antes de añadirlo
3. No comentes código eliminado
4. Los comentarios también se mantienen
5. Seguir siempre el mismo estilo
6. No dejar los comentarios para el final
7. Ser educado



Comentarios en Java - Javadoc

✓ Javadoc es una herramienta incluida en el JDK usada para generar documentación de código fuente Java en formato HTML

✓ Se requiere conocer y usar una sintaxis especial para realizar la documentación

✓ Javadoc puede ser usado desde la mayoría de IDEs profesionales

desde algún atajo a través del menú de la interfaz de usuario

✓ En Eclipse:

✓ **Project → Generate Javadoc**

Comentarios en Java - Javadoc

✓ Estos son algunas de las instrucciones más usadas en Javadoc

Etiqueta	Descripción
@author	Autor del elemento a documentar
@version	Versión del elemento de la clase
@return	Indica los parámetros de salida
@exception	Indica la excepción que puede generar
@param	Código para documentar cada uno de los parámetros
@see	Una referencia a otra clase o utilidad
@deprecated	El método ha sido reemplazado por otro

Comentarios en Java - Javadoc

- ✓ Ejemplo sobe cómo documentar un método en Java

```
/**
* Frase corta descriptiva
* Descripción del método.
* Mención al uso{@link es.loquesea.$app.util.Otra#unMetodo unMetodo}.
* @param param1 descripción del parámetro.
* @return qué devuelve el método.
* @exception tipo de excepción que lanza el método y en qué caso
* @see paquete.Clase#metodo Código al que se hace referencia
* @throws IllegalArgumentException el param1 no tiene el formato deseado
*/
```

Comentarios en Java - Javadoc

- ✓ Ejemplo sobe cómo documentar variables en Java

```
/**
* Frase corta descriptiva
* Descripción de la variable.
* Valores válidos (si aplica)
* Comportamiento en caso de que sea null(si aplica)
*/
```