1. **Refactorización**
   1. **Concepto de refactorización.**

En un desarrollo de software, hay situaciones en las que es necesario modificar partes de un software ya existente o de alguno de sus apartados o componentes. Tradicionalmente este apartado se realizaba manualmente comprobando también de forma manual donde esas modificaciones iban a obligar a cambios más o menos significativos en el resto del código.

1. **Control de versiones.**
   1. **Seguimiento de los cambios en un desarrollo.**

A lo largo del desarrollo de un proyecto de software, los archivos correspondientes al mismo experimentarán modificaciones, añadidos y distintos procesos que permiten desde la depuración de bugs hasta el añadido de nuevas funcionalidades, en un momento dado, cualquiera de las modificaciones podría dañar alguna funcionalidad, disponer de un seguimiento que permita identificar cuando se ha realizado un cambio, la naturaleza de dicho cambio y el origen del mismo, va a ser de una ayuda inestimable. El software de control de versiones es un software dedicado a esta cuestión.

* 1. **Estructura del software de control de versiones.**

1. **Documentación.**
   1. **Introducción**

Se entiende por documentación al texto escrito que acompaña a los proyectos de cualquier índole, en el caso del software es un requisito muy importante. Podemos considerar distintos tipos de documentación.

Documentación de las especificaciones.

Documentación del diseño.

Documentación del código fuente.

Documentación del usuario final.

* 1. **Documentación de las especificaciones.**

En la fase inicial de todo proyecto de software se realiza una recopilación de las distintas necesidades de este: requisitos requeridos por el cliente, requisitos funcionales del sistema desarrollado, requerimientos de hardware y software, necesidades de personal y requisitos de tiempo estimados.

Se habla de especificación de requisitos de software SRS (Software Requirements Specification) y respecto a este tipo de documentación existe una normativa recogida por la IEEE 830 (En el siguiente enlace se puede descargar una versión traducida al castellano del documento: [IEEE830-esp.doc (unican.es)](https://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf)).

En el IEEE 830 se especifica que antes de realizar un buen SRS es necesario el conocimiento de ciertas clausulas:

**Naturaleza del SRS**: Funcionalidades presentes en el software, interfaces de interacción con el mismo (bien con el usuario o bien con otro software), necesidades de tiempo de respuesta y actuación, etc.

**Ambiente del SRS**: El software contiene toda la funcionalidad o es parte un sistema mayor. Necesita definir todos los requisitos del software o hay detalles que se desarrollarán en fases posteriores.

**Características de un buen SRS**: Se requiere que sea correcto, inequívoco, completo, consistente, estable, comprobable, modificable e identificable.

**Preparación de los JOINS del SRS**: Los clientes, normalmente, no entienden bien el diseño del software y el proceso de desarrollo para escribir el SRS y los proveedores no entienden bien el problema de los clientes ni su campo de actuación. Es por lo anterior que es necesario que clientes y proveedores trabajen juntos para producir un correcto SRS.

**Evolución de los SRS**: Es necesario prever las posibles evoluciones del software, los requisitos deben especificarse completamente aún en el caso de que las posibles revisiones evolutivas se consideren inevitables (se debería especificar que tales requisitos están incompletos y será necesaria dicha revisión). En el momento de una revisión, se debe dejar constancia de esta informando de los cambios y las especificaciones que completa o ayuda a completar.

**Prototipos**: El uso de prototipos son útiles en cuanto a que permiten al cliente ver como serán las funcionalidades desplegadas por el desarrollo y poder reaccionar al mismo. Permiten anticiparse a la conducta del sistema y producir las respuestas adecuadas, así como prever nuevas respuestas a nuevas preguntas que pueden surgir durante el desarrollo. Un buen prototipo permite que el SRS sufra menos cambios durante el desarrollo permitiendo acortar los tiempos.

**Generando el diseño del SRS**: El SRS no documenta la fase de diseño, que tiene su propia documentación, pero si que debe especificar que funciones deben ser realizada, con que datos y para producir que resultados, cuáles son las situaciones de operación y para quien realiza dichas funciones.

**Requisitos del proyecto generados en el SRS**: El SRS debe dirigir el producto de software, no el proceso de producir dicho producto. Representan una compresión entre el cliente y el proveedor, así como unos compromisos de producción, que suelen ser contractuales. Se incluyen puntos importantes, como el costo, los tiempos de entrega, los métodos de desarrollo, los cumplimientos o certificaciones de calidad, la aprobación y la aceptación.

* 1. **Documentación del diseño.**

Es en esta fase en la que se decide:

La estructura de los datos, modelo de datos, diagramas entidad-relación que describen los mismos, la manera de implementarlos, etc.

Las funciones utilizadas, el contenido de las clases, sus métodos y estructuras, los atributos y las relaciones entre distintas clases y/o paquetes de clases.

El uso de librerías externas o programas de apoyo.

* 1. **Documentación del código fuente.**

Como se ha comentado numerosamente en clase, la documentación del código fuente es primordial por distintos motivos, el buen entendimiento y coordinación de los distintos grupos de trabajo, la disminución de tiempos en diferentes fases de desarrollo, la migración del proyecto a diferentes grupos de trabajo, etc.

Un programa bien documentado es más sencillo de entender y depurar en caso de aparición de errores o bugs. Su evolución también es más sencilla, los tiempos de desarrollo de nuevas implementaciones o modificaciones de las existentes se ven drásticamente reducidos cuando todo está bien documentado.

Hay que tener en cuenta que ningún programa es estático, todos los programas tienen errores que se evidenciarán antes o después y todos los programas van a sufrir modificaciones a lo largo de su vida, o por lo menos aquellos que tienen éxito.

En la documentación es importante explicar que hace una clase o método, pero también indicar el motivo por el cual se hace, se deberá indicar entre otros, de que se encarga un método, clase, paquete o variable y cuál es el uso esperado. Que algoritmo se ha implementado. También es importante indicar que es lo que queda por hacer, que habría que intentar modificar o implementar en futuras revisiones, así como problemas o limitaciones conocidas de la versión actual.

En java, disponemos de la herramienta JavaDoc que nos permite extraer la documentación incluida en el código para generar documentos en formato HTML que incluye dicha documentación relativa a la clase pertinente. La documentación que queremos que explícitamente se incluya en los documentos generados por JavaDoc se recomienda que aparezca dentro de bloques /\*\* …\*/ comenzando cada línea de documentación con un \* , ejemplo:

/\*\*

\* Esto es un ejemplo de documentación para

\* JavaDoc.

\*

\* @author Alberto Blanco

\* @version v1.0

\*

\*/

Como vemos en el bloque anterior, he usado unas etiquetas (una serie de tokens que empiezan por un carácter @) que ayudan a generar la documentación de JavaDoc.

Algunas de las etiquetas más usada son:

* @author Indica quien es el autor (programador) que ha generado la clase actual.
* @version Versión de la clase actual.
* @see Referencia a otra clase del mismo proyecto o de otro.
* @param Descripción de un parámetro.
* @return Descripción de lo que devuelve una determinada clase.
* @throws Descripción de una excepción que puede propagar, es necesaria una etiqueta throws por cada tipo de excepción.
* @deprecated Marca el método como obsoleto.
* @since Indica desde que número de versión de la clase existe el método referido.

Observación: Los comentarios de JavaDoc deben introducirse antes de las declaraciones de las clases, métodos o atributos.

Ejemplo de código documentado para su uso con JavaDoc:

/\*\*

\* <h2>Clase Empleado, se utiliza para crear y leer empleados de una BD</h2>

\*

\* Busca información de javadoc en <a href=<http://google.es>>GOOGLE</a>

\* @see <a href=<http://google.es>>GOOGLE</a>

\* @version 1-2020

\* @author ARM

\* @since 1-1-2020

\*/

public class Empleado {

/\*\*

\* Atributo Nombre de empleado

\*/

private string nombre;

/\*\*

\* Atributo Apellido de empleado

\*/

private string apellido;

/\*\*

\* Atributo Salario de empleado

\*/

private double salario;

/\*\*

\* Constructor con 3 parámetros.

\* Crea objetos empleado, con nombre, apellidos y salario.

\* @param nombre <i>Nombre del empleado</i>

\* @param apellido <i>Apellido del empleado</i>

\* @param salario <i>Salario del empleado</i>

\*/

public Empleado (String nombre, String apellido, double salario) {

this.nombre=nombre;

this.apellido=apellido;

this.salario=salario;

}

//A continuación estarían los métodos públicos y privados de la clase.

}

* 1. **Documentación del usuario final.**

Todo proyecto de software debe ir acompañado de manuales destinados a informar sobre las características y modos de utilización de las aplicaciones incluidas en el proyecto.

En la elaboración de los manuales de uso, se debe seguir ciertas normas:

* Claridad en las explicaciones, las explicaciones deben de ser claras, univocas y enfocadas al hardware y software correspondiente.
* Adecuación al usuario final, si la aplicación es de uso general hay que tener en cuenta que el usuario puede no tener conocimientos específicos sobre el uso de computadoras o software de computadores.
* Imágenes pertinentes y correspondientes a la aplicación (y versión de esta) que ayuden a la compresión de las explicaciones.
* Si las explicaciones se acompañan con ejemplos, estos deben ser lo suficientemente claros y bien documentados.
* Presentación, los manuales tienen que ser visualmente atractivos, un manual bien diseñado vende el producto y proporciona una imagen de calidad.

Se pueden considerar diferentes tipos de manuales, los principales que se pueden considerar:

* Manual de despliegue o instalación.
* Manual de usuario.
* Manual de administrador o usuario privilegiado (si lo hubiera).