



UD 04.DOCUMENTACIÓN

Entornos de desarrollo (ED)

Sergio Badal
Raúl Palao

UD 04.DOCUMENTACIÓN

4 DOCUMENTACIÓN

4.1 ¿DOCUMENTAMOS?

4.2 DOCUMENTACIÓN EXTERNA

4.3 DOCUMENTACIÓN INTERNA

4.4 HERRAMIENTA JAVADOC



4.1 ¿DOCUMENTAMOS?

- Cuando desarrollamos un proyecto se debe aportar su documentación. Esto es, **todo lo que hayamos necesitado para desarrollar el software** (código, diagramas, manuales de uso...).
- La buena documentación es esencial para un proyecto software. Sin ella un equipo se perderá en un mar de código.
- Pero, por otra parte, demasiada documentación distrae e induce error y, sobretodo, resta horas a otras fases del proyecto.
- En **ciclo en cascada** la documentación lo era todo... y en **metodologías ágiles**, la documentación debe ser mínima.



4.1 ¿DOCUMENTAMOS?

- Cuando desarrollamos un proyecto se debe aportar su documentación. Esto es, **todo lo que hayamos necesitado para desarrollar el software** (código, diagramas, manuales de uso...).
- La buena documentación es esencial para un proyecto software. Sin ella un equipo se perderá en un mar de código.
- Pero, por otra parte, demasiada documentación distrae e induce error y, sobretodo, resta horas a otras fases del proyecto.
- En **ciclo en cascada** la documentación lo era todo... y en **metodologías ágiles**, la documentación debe ser mínima.



Entonces,
¿documentamos
o no?

4.1 ¿DOCUMENTAMOS?

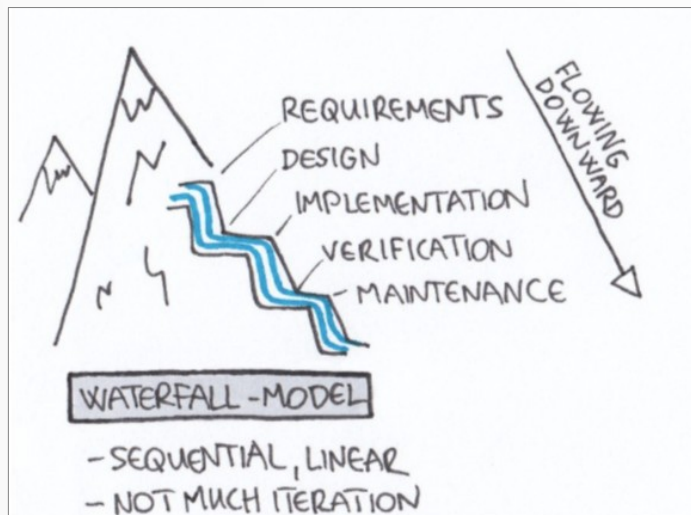
- Cuando desarrollamos un proyecto se debe aportar su documentación. Esto es, **todo lo que hayamos necesitado para desarrollar el software** (código, diagramas, manuales de uso...).
- La buena documentación es esencial para un proyecto software. Sin ella un equipo se perderá en un mar de código.
- Pero, por otra parte, demasiada documentación distrae e induce error y, sobretodo, resta horas a otras fases del proyecto.
- En **ciclo en cascada** la documentación lo era todo... y en **metodologías ágiles**, la documentación debe ser mínima.



Depende del ciclo de vida,
del tipo de proyecto, y del
tipo de cliente.

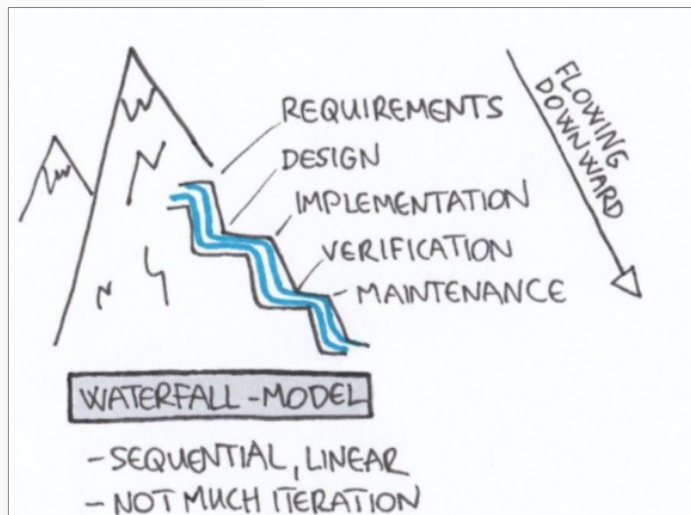
4.1 ¿DOCUMENTAMOS?

- ¿Qué dicen los ciclos de vida clásicos?
 - La documentación lo es todo.



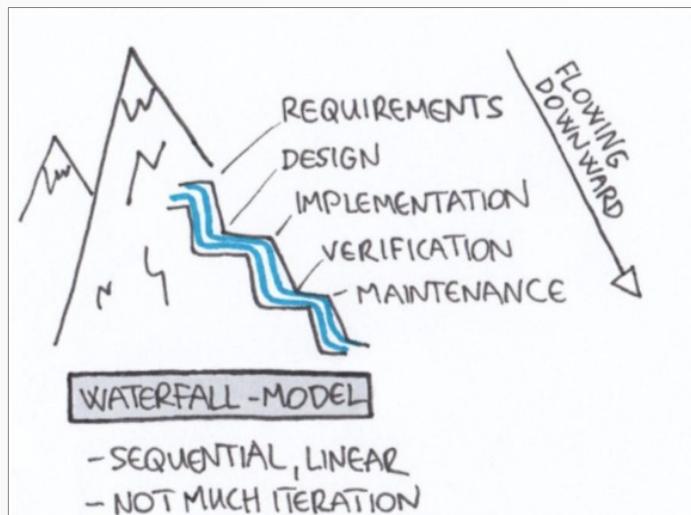
4.1 ¿DOCUMENTAMOS?

- ¿Qué dicen los ciclos de vida clásicos?
 - La documentación lo es to



4.1 ¿DOCUMENTACIÓN

- ¿Qué dicen los ciclos de vida clásicos?
 - La documentación lo es to

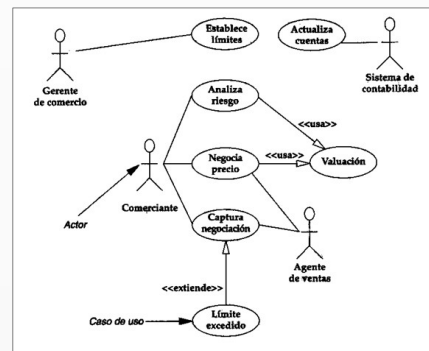
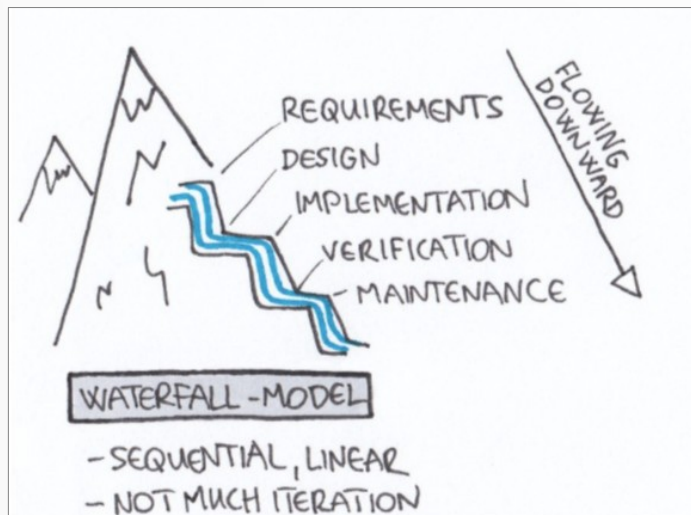
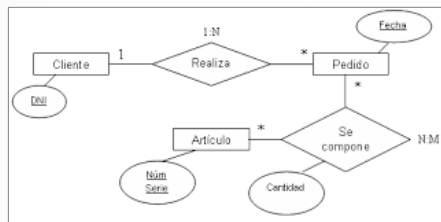
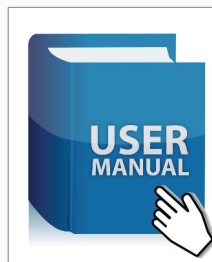


- 1.- Alcance Del Sistema
 - 1.- Planteamiento Del Problema
 - 2.- Justificación
 - 3.- Objetivos Generales Y Específicos
 - 4.- Desarrollo Con Proceso Unificado
- 2.- Análisis Y Especificación De Requisitos
 - 1.- Identificación Y Descripción De Pasos
 - 2.- Especificación De Requisitos
 - 1.- Objetivos Del Sistema
 - 2.- Requisitos De Información
 - 3.- Restricciones Del Sistema
 - 3.- Requisitos Funcionales
 - 1.- Diagramas De Casos De Uso
 - 2.- Definición De Actores
 - 3.- Documentación De Los Casos De Uso
 - 4.- Requisitos No Funcionales
- 3.- Diseño Del Sistema Xxx
 - 1.- Diagrama De Clases
 - 2.- Modelo Entidad Relación
 - 3.- Modelo Relacional
 - 4.- Diccionario De Datos
 - 5.- Diagrama De Secuencias
 - 6.- Diagrama de actividades
- 4.- Implementación
 - 1.- Arquitectura Del Sistema
 - 2.- Implementación Con Estándares
 - 3.- Arquitectura De Desarrollo
 - 4.- Estándar De Codificación
 - 5.- Sistema De Control De Versiones
 - 6.- Diagrama De Despliegue
- 5.- Pruebas
 - 1.- Planificación
 - 2.- Desarrollo De Las Pruebas
- 6.- Resultados
 - 1.- Conclusiones
 - 2.- Trabajos Futuros
 - 3.- Anexos
 - 1.- Manual De Usuario
 - 2.- Manual De Instalación

4.1 ¿DOCUMENTACIÓN

• ¿Qué dicen los ciclos de vida clásicos?

- La documentación lo es to



- 1.- Alcance Del Sistema
 - 1.- Planteamiento Del Problema
 - 2.- Justificación
 - 3.- Objetivos Generales Y Específicos
 - 4.- Desarrollo Con Proceso Unificado
- 2.- Análisis Y Especificación De Requisitos
 - 1.- Identificación Y Descripción De Pasos
 - 2.- Especificación De Requisitos
 - 1.- Objetivos Del Sistema
 - 2.- Requisitos De Información
 - 3.- Restricciones Del Sistema
 - 3.- Requisitos Funcionales
 - 1.- Diagramas De Casos De Uso
 - 2.- Definición De Actores
 - 3.- Documentación De Los Casos De Uso
 - 4.- Requisitos No Funcionales
- 3.- Diseño Del Sistema Xxx
 - 1.- Diagrama De Clases
 - 2.- Modelo Entidad Relación
 - 3.- Modelo Relacional
 - 4.- Diccionario De Datos
 - 5.- Diagrama De Secuencias
 - 6.- Diagrama de actividades
- 4.- Implementación
 - 1.- Arquitectura Del Sistema
 - 2.- Implementación Con Estándares
 - 3.- Arquitectura De Desarrollo
 - 4.- Estándar De Codificación
 - 5.- Sistema De Control De Versiones
 - 6.- Diagrama De Despliegue
- 5.- Pruebas
 - 1.- Planificación
 - 2.- Desarrollo De Las Pruebas
- 6.- Resultados
 - 1.- Conclusiones
 - 2.- Trabajos Futuros
 - 3.- Anexos
 - 1.- Manual De Usuario
 - 2.- Manual De Instalación

4.1 ¿DOCUMENTAMOS?

- ¿Qué dice el manifiesto ágil?

- Valoramos más el software que funciona que la documentación exhaustiva.

- Los documentos:

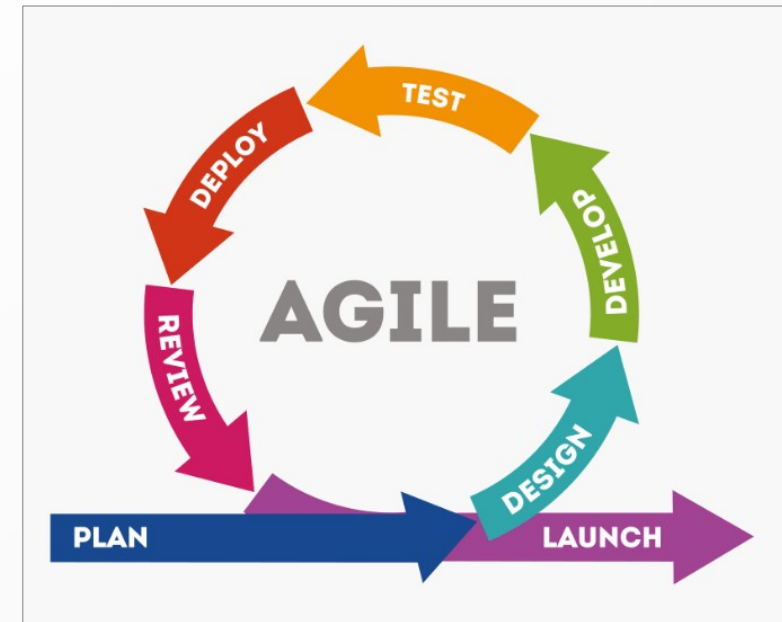
- Permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales son obligatorios.
- Su relevancia debe ser mucho menor que el producto final.

- Si la organización y los equipos se comunican a través de documentos:

- No ofrece la riqueza y generación de valor que logra la comunicación directa entre las personas
- No es comparable a la interacción con prototipos del producto.
- Ocultan la riqueza de la interacción con el producto
- Forman barreras de burocracia entre departamentos o entre personas.

- **Conclusión ÁGIL:**

- **Reducir al mínimo indispensable el uso de documentación.**
- **Solo generar la que aporte un valor directo al producto ...**
... o sea necesaria para tomar decisiones.



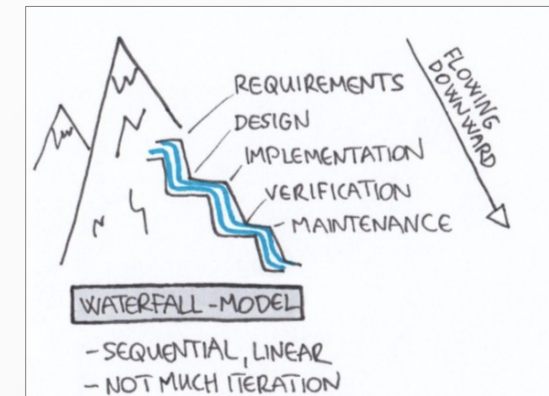
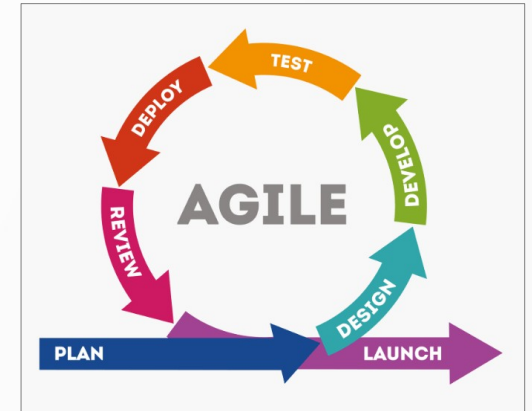
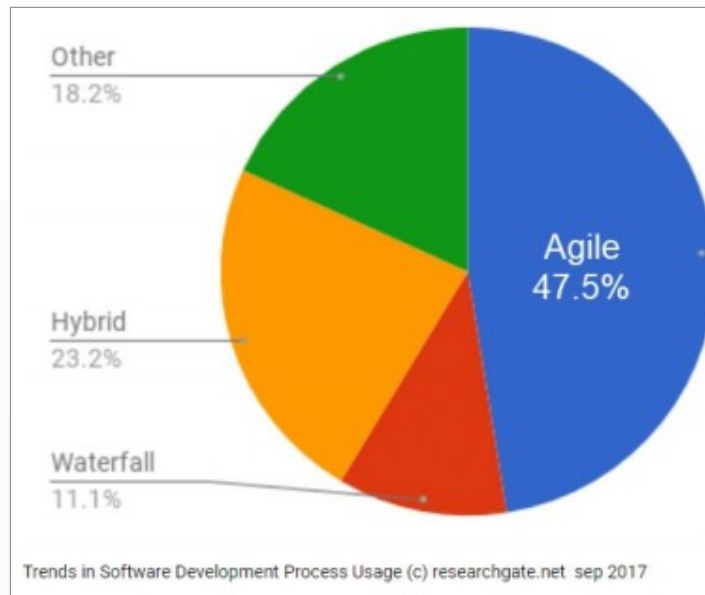
4.1 ¿DOCUMENTAMOS?

- ¿Y nosotros, qué hacemos?



4.1 ¿DOCUMENTAMOS?

- **¿Y nosotros, qué hacemos?**



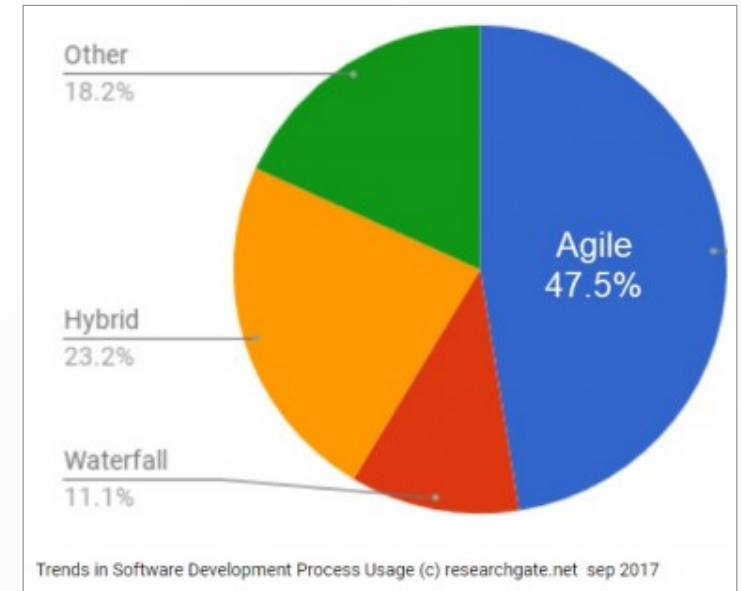
4.1 ¿DOCUMENTAMOS?

- **¿Y nosotros, qué hacemos?**

- Dado que muchas empresas siguen usando modelos clásicos...

➔ Estar preparados para ambas situaciones:

- Un contexto en cascada (clásico)
 - Un contexto ágil



UD 04.DOCUMENTACIÓN

4 DOCUMENTACIÓN

4.1 ¿DOCUMENTAMOS?

4.2 DOCUMENTACIÓN EXTERNA

4.3 DOCUMENTACIÓN INTERNA

4.4 HERRAMIENTA JAVADOC



4.2 DOCUMENTACIÓN EXTERNA

Documentación Interna

código fuente

Documentación Externa

análisis de requisitos

diseño del programa

diseño de la BD: diagrama DER

diagramas UML

Manual de instalación

Manual del usuario

Historia del desarrollo del programa

Modificaciones posteriores

4.2 DOCUMENTACIÓN EXTERNA

- El índice del documento a aportar una vez finalizado nuestro proyecto debería ser como sigue:

1.- Alcance Del Sistema

- 1.- Planteamiento Del Problema*
- 2.- Justificación*
- 3.- Objetivos Generales Y Específicos*
- 4.- Desarrollo Con Proceso Unificado*

2.- Análisis Y Especificación De Requisitos

- 1.- Identificación Y Descripción De Pasos*
- 2.- Especificación De Requisitos*

- 1.- Objetivos Del Sistema*
- 2.- Requisitos De Información*
- 3.- Restricciones Del Sistema*

3.- Requisitos Funcionales

- 1.- Diagramas De Casos De Uso*
- 2.- Definición De Actores*
- 3.- Documentación De Los Casos De Uso*
- 4.- Requisitos No Funcionales*

3.- Diseño Del Sistema Xxx

- 1.- Diagrama De Clases*
- 2.- Modelo Entidad Relación*
- 3.- Modelo Relacional*
- 4.- Diccionario De Datos*
- 5.- Diagrama De Secuencias*
- 6.- Diagrama de actividades*

4.- Implementación

- 1.- Arquitectura Del Sistema*
- 2.- Implementación Con Estándares*
- 3.- Arquitectura De Desarrollo*
- 4.- Estándar De Codificación*
- 5.- Sistema De Control De Versiones*
- 6.- Diagrama De Despliegue*

5.- Pruebas

- 1.- Planificación*
- 2.- Desarrollo De Las Pruebas*

6.- Resultados

- 1.- Conclusiones*
- 2.- Trabajos Futuros*
- 3.- Anexos*
 - 1.- Manual De Usuario*
 - 2.- Manual De Instalación*

4.2 DOCUMENTACIÓN EXTERNA

- El índice del documento a aportar una vez finalizado nuestro proyecto debería ser como sigue:

- 1.- Alcance Del Sistema
 - 1.- Planteamiento Del Problema
 - 2.- Justificación
 - 3.- Objetivos Generales Y Específicos
 - 4.- Desarrollo Con Proceso Unificado
- 2.- Análisis Y Especificación De Requisitos
 - 1.- Identificación Y Descripción De Pasos
 - 2.- Especificación De Requisitos
 - 1.- Objetivos Del Sistema
 - 2.- Requisitos De Información
 - 3.- Restricciones Del Sistema
- 3.- Requisitos Funcionales
 - 1.- Diagramas De Casos De Uso
 - 2.- Definición De Actores
 - 3.- Documentación De Los Casos De Uso
- 4.- Requisitos No Funcionales

Requerimientos Funcionales

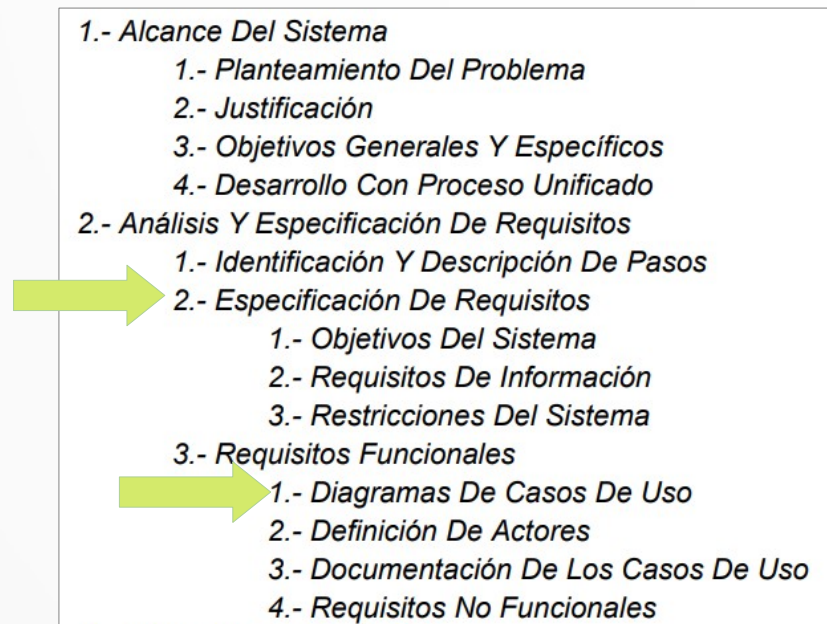
Los requisitos funcionales son declaraciones de los **servicios que prestará** el sistema, en la forma en que reaccionará a determinados insumos. Cuando hablamos de las entradas, no necesariamente hablamos sólo de las entradas de los usuarios. Pueden ser interacciones con otros sistemas, respuestas automáticas, procesos predefinidos. En algunos casos, los requisitos funcionales de los sistemas también establecen explícitamente lo que el **sistema no debe hacer**. Es importante recordar esto

Requisitos no funcionales

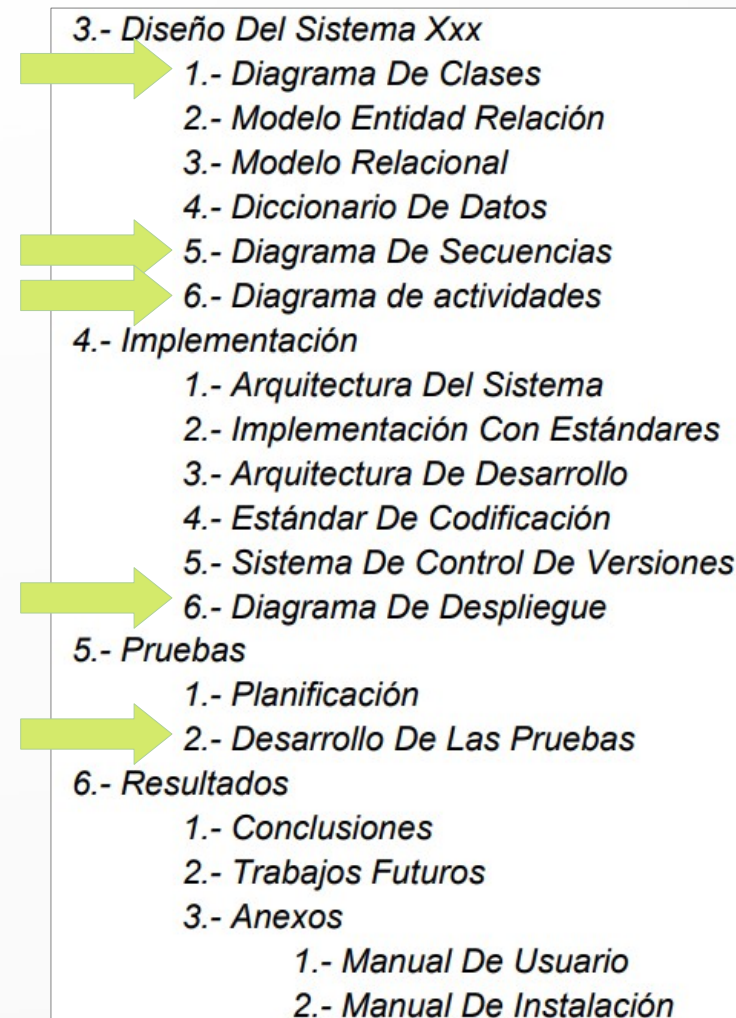
Se trata de requisitos que no se refieren directamente a las funciones específicas suministradas por el sistema (características de usuario), sino a las propiedades del sistema: rendimiento, seguridad, disponibilidad. En palabras más sencillas, no hablan de “lo que” hace el sistema, sino de “cómo” lo hace. Alternativamente, definen restricciones del sistema tales como la capacidad de los dispositivos de entrada/salida y la representación de los datos utilizados en la interfaz del sistema.

4.2 DOCUMENTACIÓN EXTERNA

- El índice del documento a aportar una vez finalizado nuestro proyecto debería ser como sigue:



VISTO/POR VER EN
ENTORNOS DE
DESARROLLO



UD 04.DOCUMENTACIÓN

4 DOCUMENTACIÓN

4.1 ¿DOCUMENTAMOS?

4.2 DOCUMENTACIÓN EXTERNA

4.3 DOCUMENTACIÓN INTERNA

4.4 HERRAMIENTA JAVADOC



4.3 DOCUMENTACIÓN INTERNA

Documentación Interna

código fuente

Documentación Externa

análisis de requisitos

diseño del programa

diseño de la BD: diagrama DER

diagramas UML

Manual de instalación

Manual del usuario

Historia del desarrollo del programa

Modificaciones posteriores

4.3 DOCUMENTACIÓN INTERNA

- Una vez que se genera el código fuente, la función de un módulo debe resultar clara sin necesidad de referirse a ninguna especificación del diseño.
 - En otras palabras:
 - El código debe ser, POR SÍ SOLO, comprensible.
 - El buen código (**CÓDIGO LIMPIO**) se comenta solo.
 - Esto quiere decir que no es necesario comentar todas las líneas.
- Sin embargo, sí es necesario comentar algunas partes del código.
- La posibilidad de expresar comentarios en **lenguaje natural** como parte del listado del código fuente es algo que aparece en todos los lenguajes de propósito general.
- Los comentarios pueden resultar una clara guía durante la última fase de la ingeniería del software, el mantenimiento.



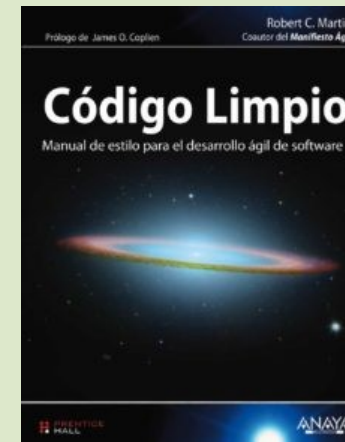
4.3 DOCUMENTACIÓN INTERNA

- Una vez que s
debe resultar
- En otra
- El
- El
- Es
- Sin embargo,
- La posibilidad
aparece en t
- Los comenta
mantenimier

Consejos para un código limpio:

- 1) Evitar el uso de complicadas comparaciones condicionales
- 2) Eliminar las comparaciones con condiciones negativas
- 3) Evitar un gran anidamiento de bucles o de condiciones
- 4) Usar paréntesis para clarificar las expresiones lógicas o aritméticas
- 5) Usar espacios y/o símbolos claros para aumentar la legibilidad

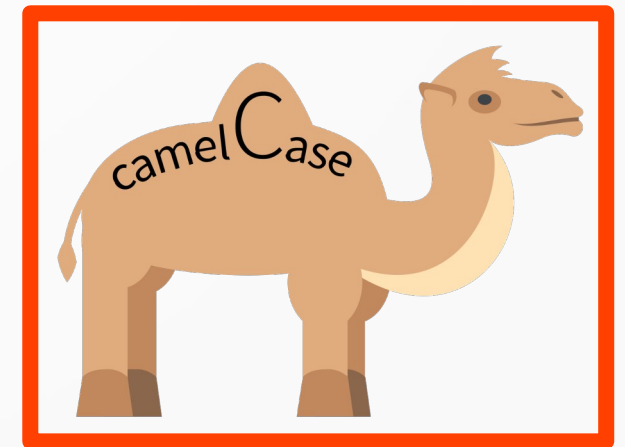
6) Pensar "¿Podría yo entender esto si no fuera la persona que lo codificó?"



ón y
niento

4.3 DOCUMENTACIÓN INTERNA

- ¿Cómo, y cuándo, comentar el código?
 - Buscar una correcta organización visual del programa también es *comentar el código*
... que no muerda, ni sea una sábana eterna ...
 - La elección de nombres de identificadores(*) significativos es crucial para la legibilidad
... en inglés, en castellano, camelCase, lowercase ...
 - En lenguajes que limitan la longitud de los identificadores los comentarios son VITALES.
 - Al principio de cada módulo debe haber un **comentario de prólogo**



(*) los identificadores son las variables,
funciones, métodos, clases, paquetes, etiquetas...

4.3 DOCUMENTACIÓN INTERNA

- **Comentario de prólogo**

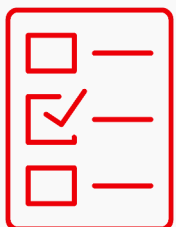
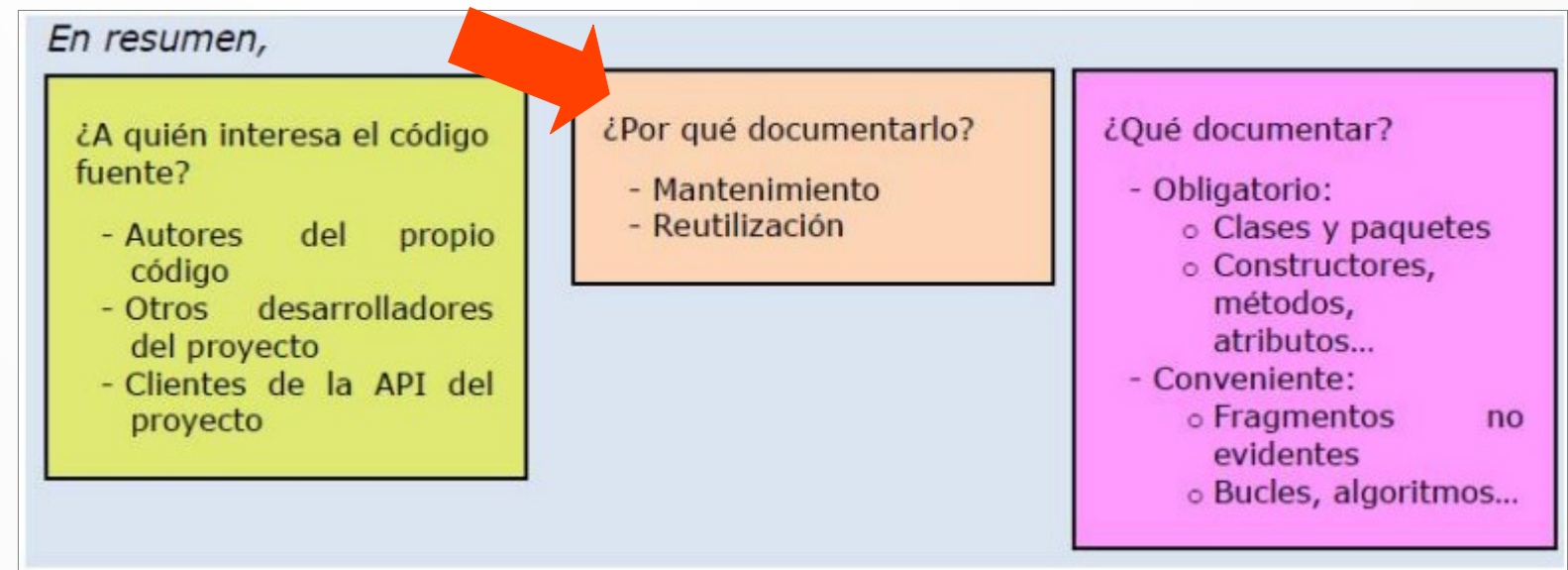
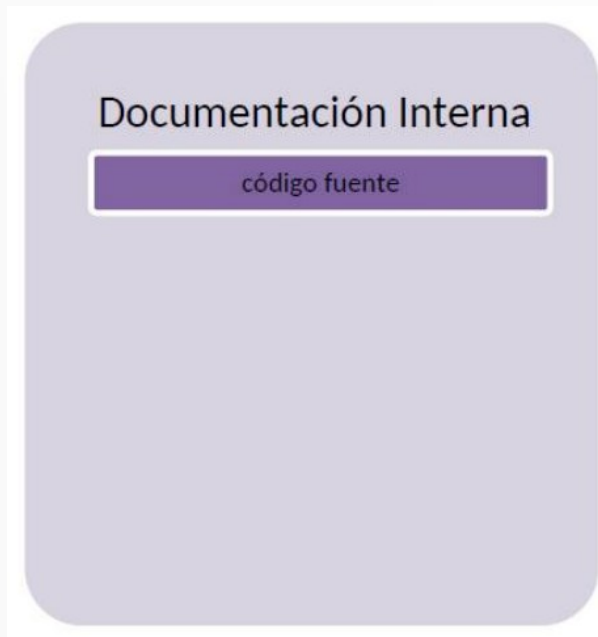
- Al principio de cada módulo debe haber un comentario de prólogo que indique el nombre de la aplicación/módulo/fichero, resuma su función y describa su autor.
- Se propone la siguiente estructura (igual de válida de muchas otras):
 1. Una sentencia que indique la función del módulo.
 2. Una descripción de la interfaz:
 - a. un ejemplo de “secuencia de llamada”
 - b. una descripción de todos los argumentos
 - c. una lista de los módulos subordinados
 3. Una explicación de los datos pertinentes, tales como las variables importantes y su uso, restricciones y limitaciones y de otra información importante
 4. Una historia del desarrollo que incluya:
 - a. el diseñador del módulo (autor)
 - b. el revisor (auditor) y la fecha
 - c. fechas de modificación

```
/**
 * <h2> Clase Empleado, se utiliza para crear y leer empleados de una BD </h2>
 *
 * Busca información de javadoc en <a href=http://google.com>GOOGLE</a>
 * @see <a href=http://www.google.com>Google</a>
 * @version 1-2014
 * @author ARM * @since 1-1-2014
 */
public class Empleado{

/**
 * Constructor con 3 parámetros
 * Crea objetos empleado, con nombre, apellidos y salario
 * @param nombre Nombre del empleado
 * @param apellido Apellido del empleado
 * @param salario Salario del empleado
 * public Empleado(String nombre, String apellido, double salario){
 *     this.nombre=nombre;
 *     this.apellido=apellido;
 *     this.salario=salario;
 * }
}
```

(*) los identificadores son las variables, funciones, métodos, clases, paquetes, etiquetas...

4.3 DOCUMENTACIÓN INTERNA



¡PREGUNTA CASI
SEGURA DEL
EXAMEN!

UD 04.DOCUMENTACIÓN

4 DOCUMENTACIÓN

4.1 ¿DOCUMENTAMOS?

4.2 DOCUMENTACIÓN EXTERNA

4.3 DOCUMENTACIÓN INTERNA

4.4 HERRAMIENTA JAVADOC



4.4 HERRAMIENTA JAVADOC

- **HERRAMIENTA: JAVADOC**

- Javadoc es

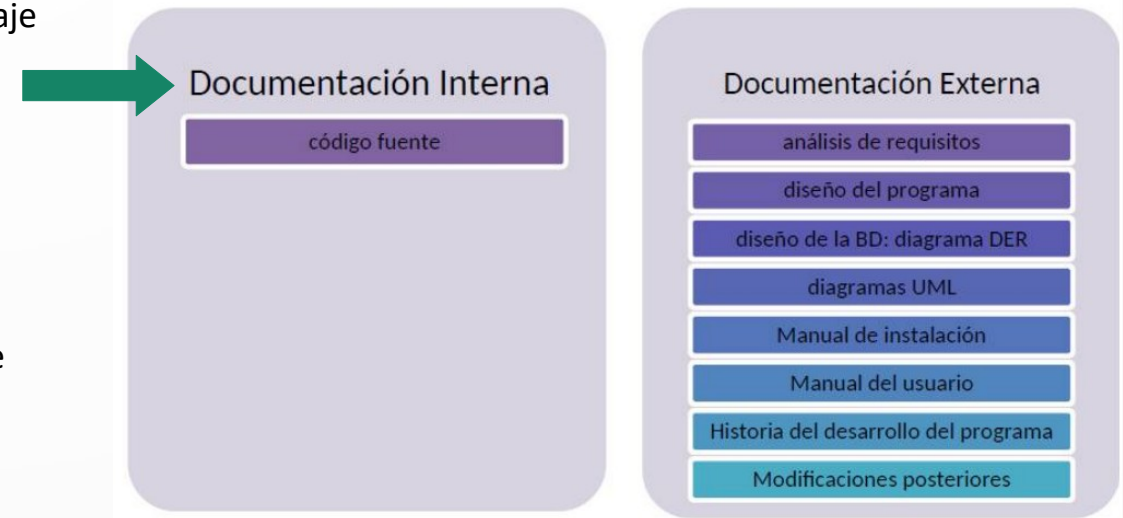
... herramienta/módulo/plugin/**aplicación/programa** para el lenguaje java, que permite generar la documentación de la interfaz o API (Application Programming Interface) de los **programas** desarrollados.

- **¿Qué hace?**

- Analiza las declaraciones y los comentarios del código fuente
- Produce páginas html que ...
... describen clases, clases internas (subclases), interfaces, constructores y atributos de la clase (campos).

- **¿Para qué sirve?**

- Generamos documentación interna **dentro del mismo código**



4.4 HERRAMIENTA JAVADOC

Generated Documentation x

file:///D:/Personal/Proyectos/Proyectos%20SE/JavaDoc/JavaDoc/doc/index.html

MiEntelV3 Catalogo Entel Paginas Sociales Java Web Android Seguridad Movil Aeropuerto Arduino

All Classes

[Principal](#)

classes

Class Principal

java.lang.Object
└─ classes.Principal

```
public class Principal
extends java.lang.Object
```

Principal.java Representa la clase principal del proyecto.

Version:
1.0

Author:
.

Constructor Summary

[Principal\(\)](#)
Constructor para la clase Principal, donde se inicializan las variables.

Method Summary


java.lang.String	getMensaje() Se obtiene el valor del mensaje.
static void	main (java.lang.String[] arg) Metodo MAIN que realiza el llamado al constructor de la clase.
void	

4.4 HERRAMIENTA JAVADOC

Los comentarios en Java pueden ser de una línea, de varias o Javadoc. Los Javadoc son como los multilínea pero comienzan con 2 asteriscos: `/**` Una línea

```
/*
 * Comentario Multilínea
 */

/**
 * Comentario Javadoc
 */
```



Un detalle importante a tener en cuenta es que SIEMPRE que se quiera comentar algo, una clase, un método, una variable, etc..., dicho comentario se debe poner inmediatamente antes del ítem a comentar. En caso contrario la herramienta de generación automática no lo reconocerá.

Los comentarios javadoc tienen dos partes:

- La parte de la descripción
- La parte de etiquetas o *tags*

Ejemplo:

```
/**
 *
 * Descripción principal ( Texto / HTML )
 *
 * Tags ( Texto / HTML )
 */
```

4.4 HERRAMIENTA JAVADOC

```
/**
 * Returns the index of the first occurrence of the specified element in
 * this vector, searching forwards from <code>index</code>, or returns -1 if
 * the element is not found.
 *
 * @param o element to search for
 * @param index index to start searching from
 * @return the index of the first occurrence of the element in
 *         this vector at position <code>index</code> or later in the vector;
 *         <code>-1</code> if the element is not found
 * @throws IndexOutOfBoundsException if the specified index is negative
 * @see    Object#equals(Object)
 */
public int indexOf(Object o, int index) ...
```

indexOf

```
public int indexOf(Object o,
                  int index)
```

Returns the index of the first occurrence of the specified element in this vector, searching forwards from `index`, or returns -1 if the element is not found.

Parameters:

`o` - element to search for
`index` - index to start searching from

Returns:

the index of the first occurrence of the element in this vector at position `index` or later in the vector; -1 if the element is not found.

Throws:

[IndexOutOfBoundsException](#) - if the specified index is negative

See Also:

[Object.equals\(Object\)](#)

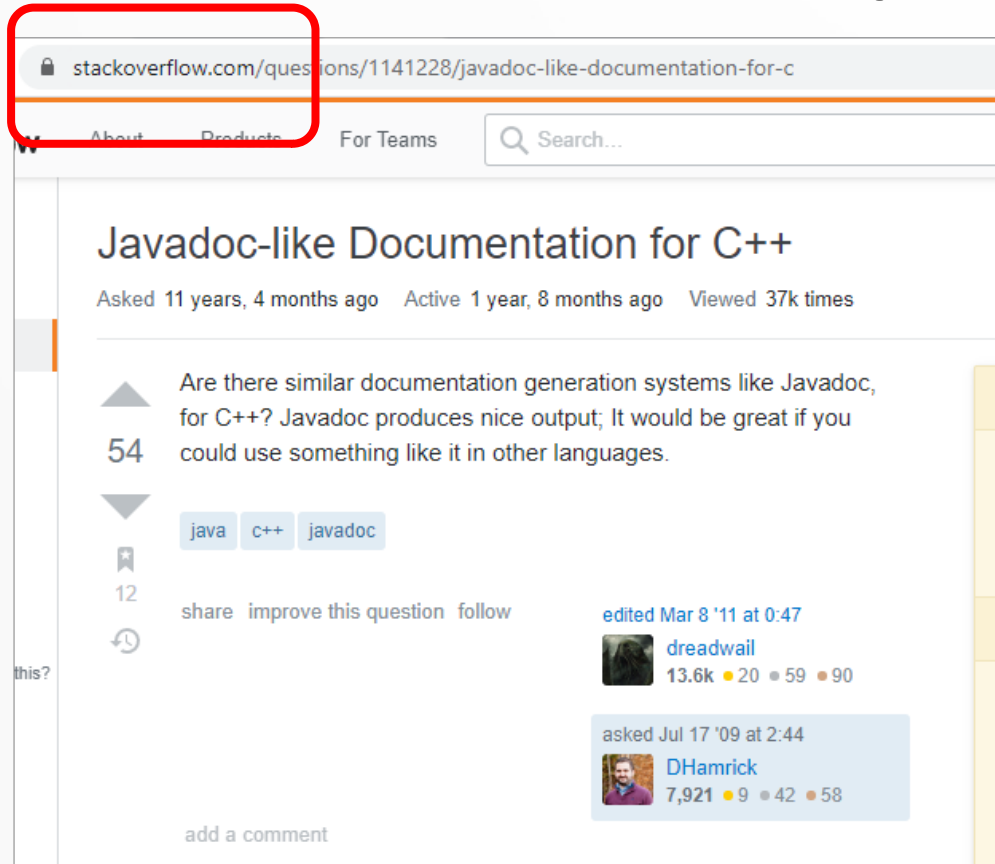
4.4 HERRAMIENTA JAVADOC

- ¿Y si no programo en Java? ¿Hay vida más allá de Java?

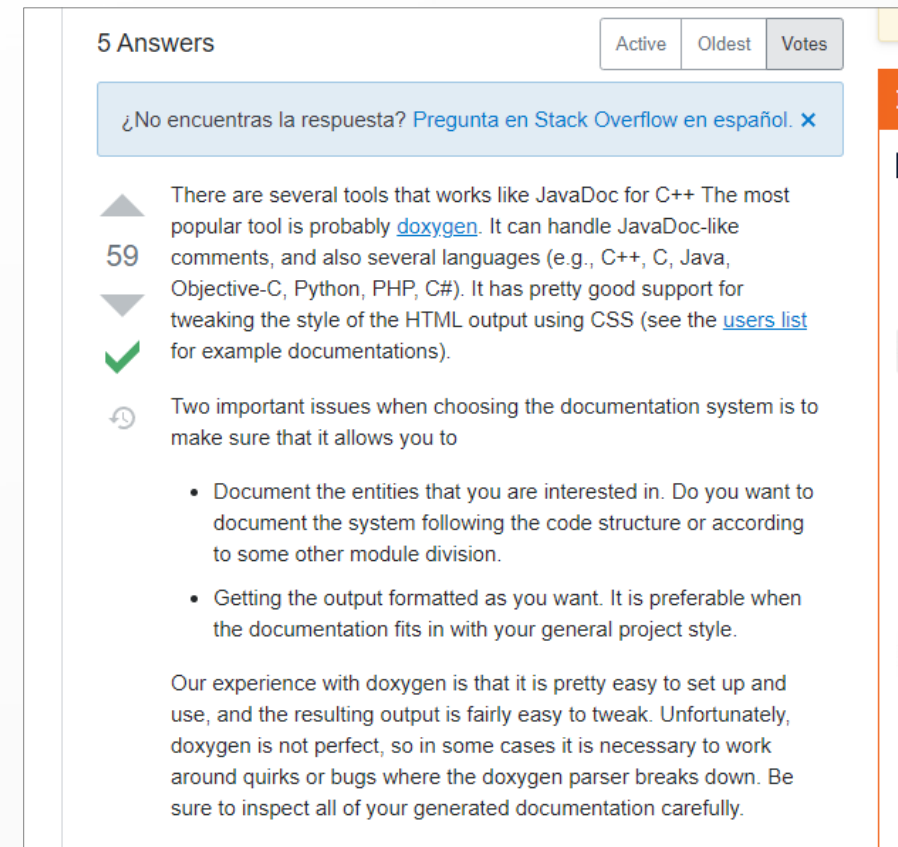


4.4 HERRAMIENTA JAVADOC

- ¿Qué dice la Comunidad (de desarrolladores)?



The screenshot shows a Stack Overflow question page. The URL bar at the top is highlighted with a red rectangle and contains the text: `stackoverflow.com/questions/1141228/javadoc-like-documentation-for-c`. The question title is "Javadoc-like Documentation for C++". Below the title, it says "Asked 11 years, 4 months ago", "Active 1 year, 8 months ago", and "Viewed 37k times". The question text is: "Are there similar documentation generation systems like Javadoc, for C++? Javadoc produces nice output; It would be great if you could use something like it in other languages." There are 54 votes and 12 answers. The question is tagged with "java", "c++", and "javadoc". The user "DHamrick" is the asker, with a reputation of 7,921. The user "dreadwail" is the top answerer, with a reputation of 13.6k.



The screenshot shows the answers section of the Stack Overflow question. It displays "5 Answers". The first answer is by "dreadwail" and has 59 votes. The answer text is: "There are several tools that works like JavaDoc for C++ The most popular tool is probably [doxygen](#). It can handle JavaDoc-like comments, and also several languages (e.g., C++, C, Java, Objective-C, Python, PHP, C#). It has pretty good support for tweaking the style of the HTML output using CSS (see the [users list](#) for example documentations)." The second answer is by "DHamrick" and has 58 votes. The answer text is: "Two important issues when choosing the documentation system is to make sure that it allows you to" followed by two bullet points: "• Document the entities that you are interested in. Do you want to document the system following the code structure or according to some other module division." and "• Getting the output formatted as you want. It is preferable when the documentation fits in with your general project style." The answer concludes with: "Our experience with doxygen is that it is pretty easy to set up and use, and the resulting output is fairly easy to tweak. Unfortunately, doxygen is not perfect, so in some cases it is necessary to work around quirks or bugs where the doxygen parser breaks down. Be sure to inspect all of your generated documentation carefully."

4.4 HERRAMIENTA JAVADOC

Doxygen

Generate documentation from source code

Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL and to some extent D.

Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in \LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.
2. You can [configure](#) doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams which are all generated automatically.
3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are available.

