



UD 05.DOCUMENTACIÓN

v1.1 09.04.21

ceedcv
CENTRE ESPECÍFIC
D'EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

Entornos de desarrollo (ED)

Sergio Badal

Carlos Espinosa

c.espinosamoreno@edu.gva.es

Extraído de los apuntes de:

Paco Aldarias; Cristina Álvarez Villanueva; Fco. Javier Valero Garzón; M.^a Carmen Safont



UD 05.DOCUMENTACIÓN

- **Pasos a seguir**

- 1) Lee la documentación (PDF)
- 2) Instala el software necesario (sigue los pasos)
- 3) Realiza los TESTS todas las veces que quieras
- 4) Acude al FORO DE LA UNIDAD
 - Para cualquier duda sobre esta unidad
- 5) Acude al FORO DEL MÓDULO
 - Para cualquier duda sobre el módulo



UD 05.DOCUMENTACIÓN

- ¿Qué veremos en esta UNIDAD?
 - SEMANA ÚNICA
 - DOCUMENTACIÓN
 - JAVADOC en NETBEANS/ECLIPSE
 - PRÁCTICA NO EVALUABLE SOBRE JAVADOC



UD 05.DOCUMENTACIÓN

- ¿Qué veremos en esta UNIDAD?
 - SEMANA ÚNICA
 - DOCUMENTACIÓN
 - JAVADOC en NETBEANS/ECLIPSE
 - PRÁCTICA NO EVALUABLE SOBRE JAVADOC



Finalmente, no habrá más prácticas evaluables pero, como mínimo, dos preguntas de la parte de test del examen de evaluación serán de esta unidad.

UD 05.DOCUMENTACIÓN

5 DOCUMENTACIÓN

5.1 ¿DOCUMENTAMOS?

5.2 DOCUMENTACIÓN EXTERNA

5.3 DOCUMENTACIÓN INTERNA

5.4 HERRAMIENTA JAVADOC

- PRÁCTICA NO EVALUABLE SOBRE JAVADOC



5.1 ¿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.



5.1 ¿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?



5.1 ¿DOCUMENTAMOS?

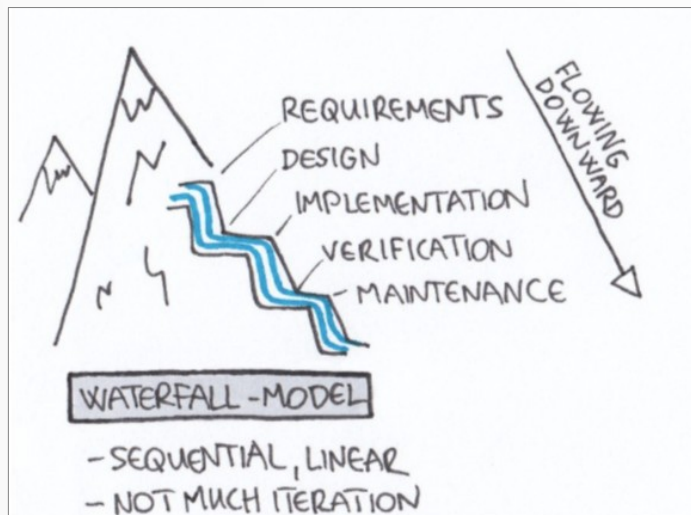
- Cuando desarrollamos un proyecto se debe aportar su documentación. Esto es, **todo lo que hayamos necesitado para desarrollar el software** (código, diagramas, manuales de uso...).
- La buena documentación es esencial para un proyecto software. Sin ella un equipo se perderá en un mar de código.
- Pero, 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.

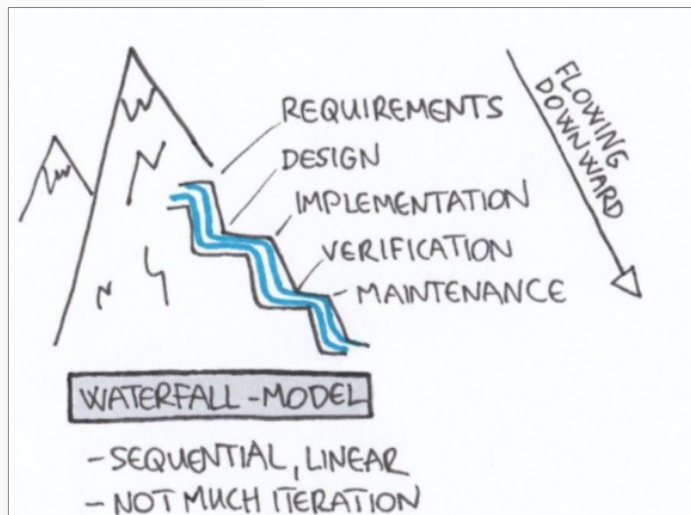
5.1 ¿DOCUMENTAMOS?

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



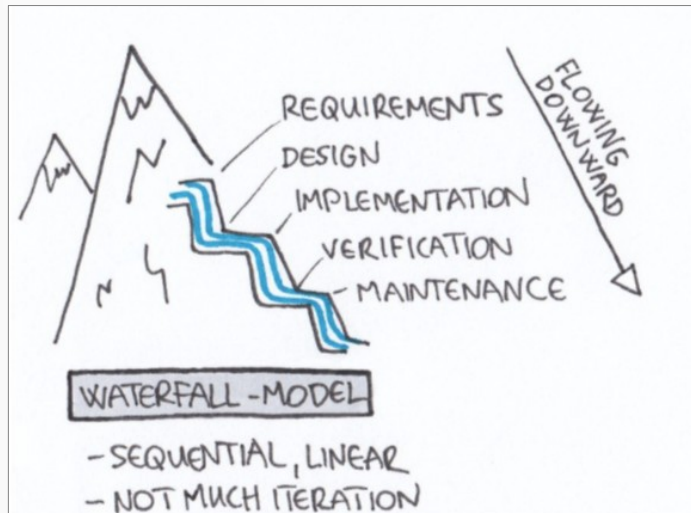
5.1 ¿DOCUMENTAMOS?

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



5.1 ¿DOCUMENTAMOS?

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

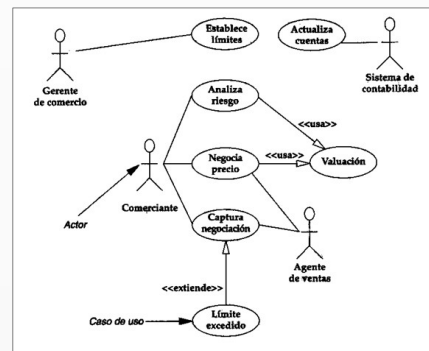
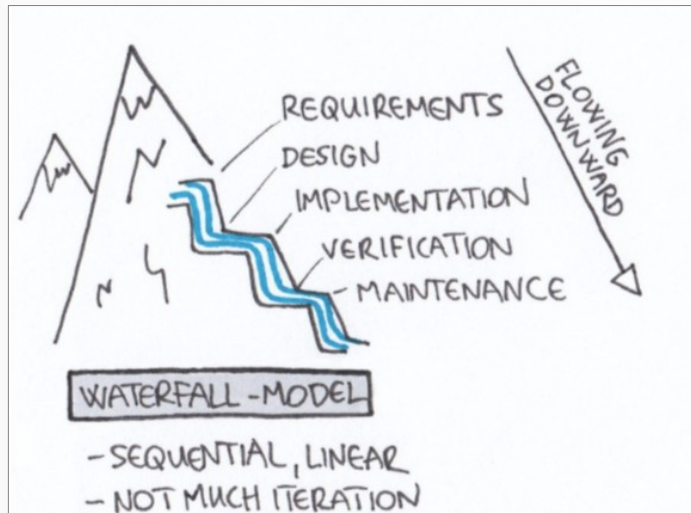
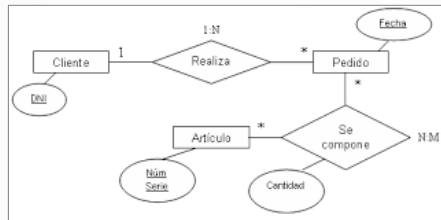
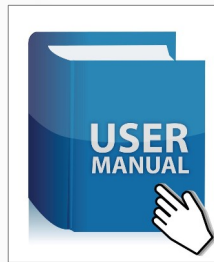


- 1.- Alcance Del Sistema
 - 1.- Planteamiento Del Problema
 - 2.- Justificación
 - 3.- Objetivos Generales Y Específicos
 - 4.- Desarrollo Con Proceso Unificado
- 2.- Análisis Y Especificación De Requisitos
 - 1.- Identificación Y Descripción De Pasos
 - 2.- Especificación De Requisitos
 - 1.- Objetivos 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

5.1 ¿DOCUMENTAMOS?

• ¿Qué dicen los ciclos de vida clásicos?

- La documentación lo es todo.



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

5.1 ¿DOCUMENTAMOS?

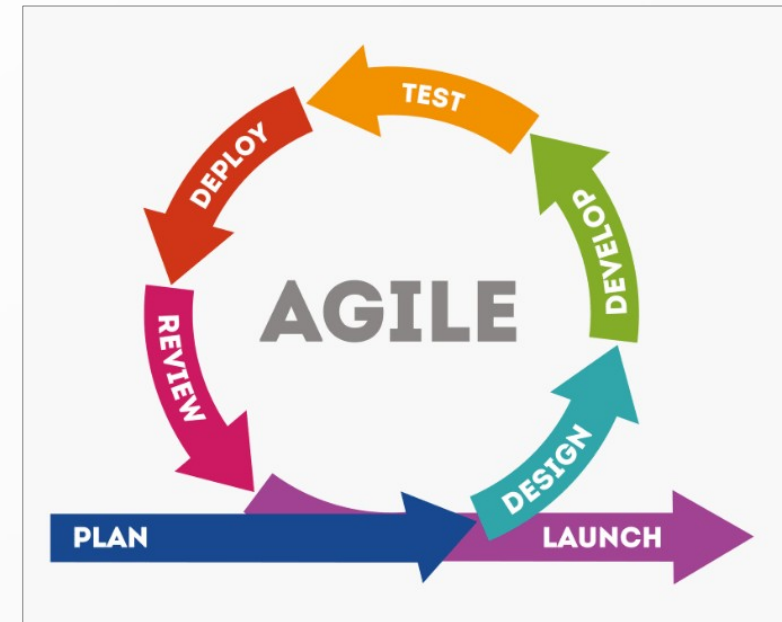
- **¿Qué dice el manifiesto ágil?**

- **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.



5.1 ¿DOCUMENTAMOS?

- **¿Qué dice la Comunidad (de desarrolladores)?**

Is it wrong not to create Javadoc for my code?

Asked 9 years, 5 months ago Active 9 years, 4 months ago Viewed 7k times



8



I do a lot of Java programming at my work (I'm an intern) and I was wondering if it is generally a rule to create javadoc to accompany my code. I usually document every method and class anyways, but I find it hard to adhere to Javadoc's syntax (writing down the variables and the output so that the parser can generate html).

I've looked at a lot of C programming and even C++ and I like the way they are commented. Is it wrong not to supply javadoc with my code?

5.1 ¿DOCUMENTAMOS?

- ¿Qué dice la Comunidad (de desarrolladores)?

Is it wrong not to create Javadoc for my code?

Asked 9 years, 5 months ago · Active 9 years, 4 months ago · Viewed 7k times



8



I do a lot of Java programming at my work (I'm an intern) and I was wondering if it is generally a rule to create javadoc to accompany my code. I usually document every method and class anyways, but I find it hard to adhere to Javadoc's syntax (writing down the variables and the output so that the parser can generate html).

I've looked at a lot of C programming and even C++ and I like the way they are commented. Is it wrong not to supply javadoc with my code?

I would go as far as saying that if your method/class *requires* Javadoc, you should rethink naming/organization. One should aim to *need* as little documentation in code as possible. If you really need documentation, Javadoc is *the* way to go, but always try to make code such that it is as self explanatory as possible. So definitely, no Javadoc for everything, unless it's needed which then means that you've got really ugly piece of code in your hands. – [merryprankster](#) Jul 22 '11 at 9:23

@merryprankster, I'm coming in late to this, but your first assertion goes somewhat too far. In my lifetime I've seen plenty of 100% understandable and otherwise elegantly and clearly written code that *still* benefits from documentation, javadoc or otherwise. I remember once someone from my distant past saying something similarly broken: "If your code *needs* comments, then redesign your code." That was a mistake as well. – [tgm1024--Monica was mistreated](#) Oct 27 '18 at 13:09

5.1 ¿DOCUMENTAMOS?

- ¿Qué dice la Comunidad (de desarrolladores)?

Is it wrong not to create Javadoc for my code?

Asked 9 years, 5 months ago · Active 9 years, 4 months ago · Viewed 7k times

8
I do a lot of Java programming at my work (I'm an intern) and I was wondering if it is generally a rule to create javadoc to accompany my code. I usually document every method and class anyways, but I find it hard to adhere to Javadoc's syntax (writing down the variables and the output so that the parser can generate html).

I've looked at a lot of C programming and even C++ and I like the way they are commented. Is it wrong not to supply javadoc with my code?

I would go as far as saying that if your method/class *requires* Javadoc, you should rethink naming/organization. One should aim to *need* as little documentation in code as possible. If you really need documentation, Javadoc is *the* way to go, but always try to make code such that it is as self explanatory as possible. So definitely, no Javadoc for everything, unless it's needed which then means that you've got really ugly piece of code in your hands. – [merryprankster](#) Jul 22 '11 at 9:23

@merryprankster, I'm coming in late to this, but your first assertion goes somewhat too far. In my lifetime I've seen plenty of 100% understandable and otherwise elegantly and clearly written code that *still* benefits from documentation, javadoc or otherwise. I remember once someone from my distant past saying something similarly broken: "If your code *needs* comments, then redesign your code." That was a mistake as well. – [tgm1024--Monica was mistreated](#) Oct 27 '18 at 13:09

@merryprankster, Boiling it down: I think if I were to rephrase your question to match my beliefs (for what it's worth (zero)), it might be more akin to: "In all programming, always keep an eye out for how much of your documentation can be lessened by a cleaner approach in the code. Excess documentation cannot make up for unnecessary complexity." But of course, that's my view on things. – [tgm1024--Monica was mistreated](#) Nov 6 '18 at 22:41

5.1 ¿DOCUMENTAMOS?

- ¿Qué dice la Comunidad (de desarrolladores)?

Is it wrong not to create Javadoc for my code?

Asked 9 years, 5 months ago · Active 9 years, 4 months ago · Viewed 7k times

8
I do a lot of Java programming at my work (I'm an intern) and I was wondering if it is generally a rule to create javadoc to accompany my code. I usually document every method and class anyways, but I find it hard to adhere to Javadoc's syntax (writing down the variables and the output so that the parser can generate html).

I've looked at a lot of C programming and even C++ and I like the way they are commented. Is it wrong not to supply javadoc with my code?

I would go as far as saying that if your method/class *requires* JavaDoc, you should rethink naming/organization. One should aim to *need* as little documentation in code as possible. If you really need documentation, JavaDoc is *the* way to go, but always try to make code such that it is as self explanatory as possible. So definitely, no JavaDoc for everything, unless it's needed which then means that you've got really ugly piece of code in your hands. – merryprankster Jul 22 '11 at 9:23

@merryprankster, I'm coming in late to this, but your first assertion goes somewhat too far. In my lifetime I've seen plenty of 100% understandable and otherwise elegantly and clearly written code that *still* benefits from documentation, javadoc or otherwise. I remember once someone from my distant past saying something similarly broken: "If your code *needs* comments, then redesign your code." That was a mistake as well. – tgm1024--Monica was mistreated Oct 27 '18 at 13:09

@merryprankster, Boiling it down: I think if I were to rephrase your question to match my beliefs (for what it's worth (zero)), it might be more akin to: "In all programming, always keep an eye out for how much of your documentation can be lessened by a cleaner approach in the code. Excess documentation cannot make up for unnecessary complexity." But of course, that's my view on things. – tgm1024--Monica was mistreated Nov 6 '18 at 22:41

[Ver hilo en STACKOVERFLOW](#)

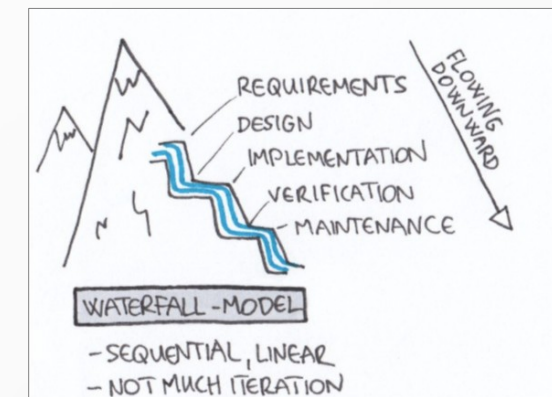
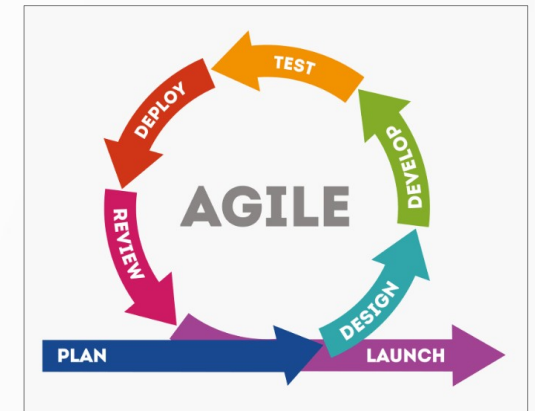
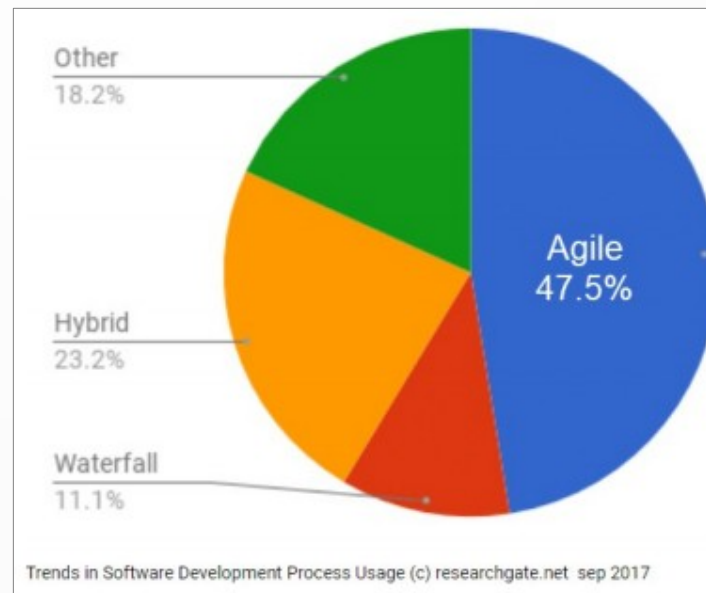
5.1 ¿DOCUMENTAMOS?

- **¿Y nosotros, qué hacemos?**



5.1 ¿DOCUMENTAMOS?

- ¿Y nosotros, qué hacemos?



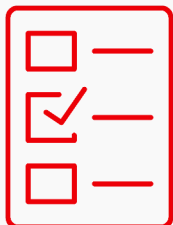
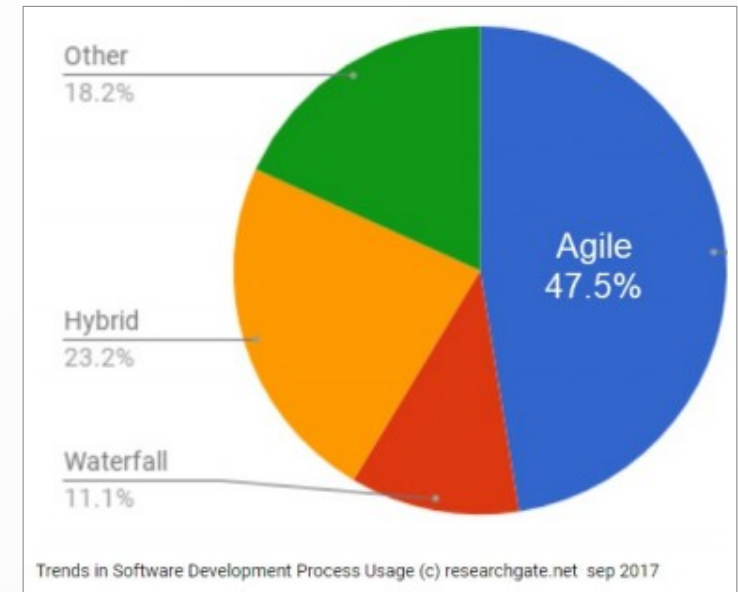
5.1 ¿DOCUMENTAMOS?

- **¿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



¡PREGUNTA CASI
SEGURA DEL EXAMEN!

UD 05.DOCUMENTACIÓN

5 DOCUMENTACIÓN

5.1 ¿DOCUMENTAMOS?

5.2 DOCUMENTACIÓN EXTERNA

5.3 DOCUMENTACIÓN INTERNA

5.4 HERRAMIENTA JAVADOC

- PRÁCTICA NO EVALUABLE SOBRE JAVADOC



5.2 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

5.2 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*

5.2 DOCUMENTACIÓN EXTERNA

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

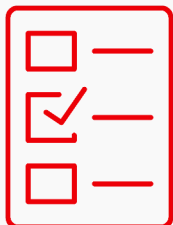
- 1.- Alcance Del Sistema
 - 1.- Planteamiento Del Problema
 - 2.- Justificación
 - 3.- Objetivos Generales Y Específicos
 - 4.- Desarrollo Con Proceso Unificado
- 2.- Análisis Y Especificación De Requisitos
 - 1.- Identificación Y Descripción De Pasos
 - 2.- Especificación De Requisitos
 - 1.- Objetivos 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

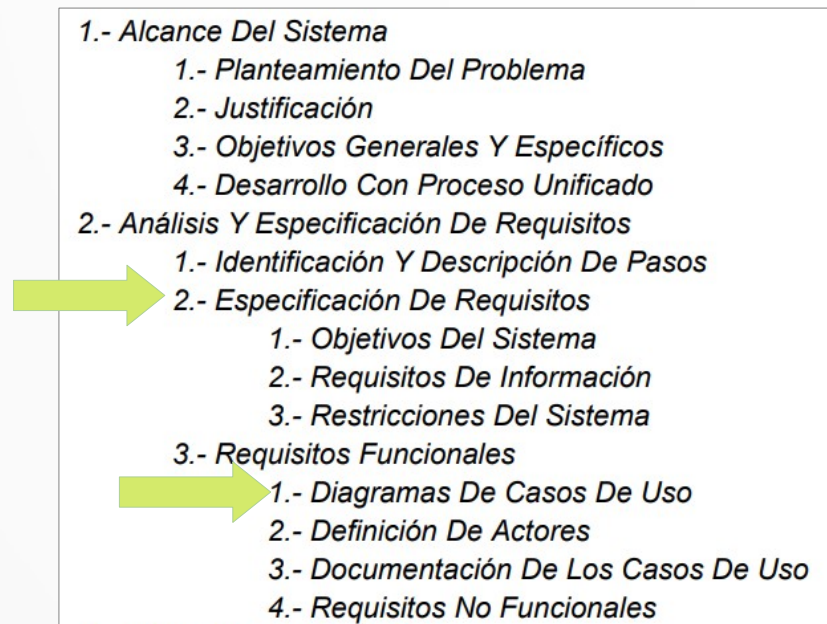
Se trata de requisitos que no se refieren directamente a las funciones específicas suministradas por el sistema (características de usuario), sino a las propiedades del sistema: rendimiento, seguridad, disponibilidad. En palabras más sencillas, no hablan de “lo que” hace el sistema, sino de “cómo” lo 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.



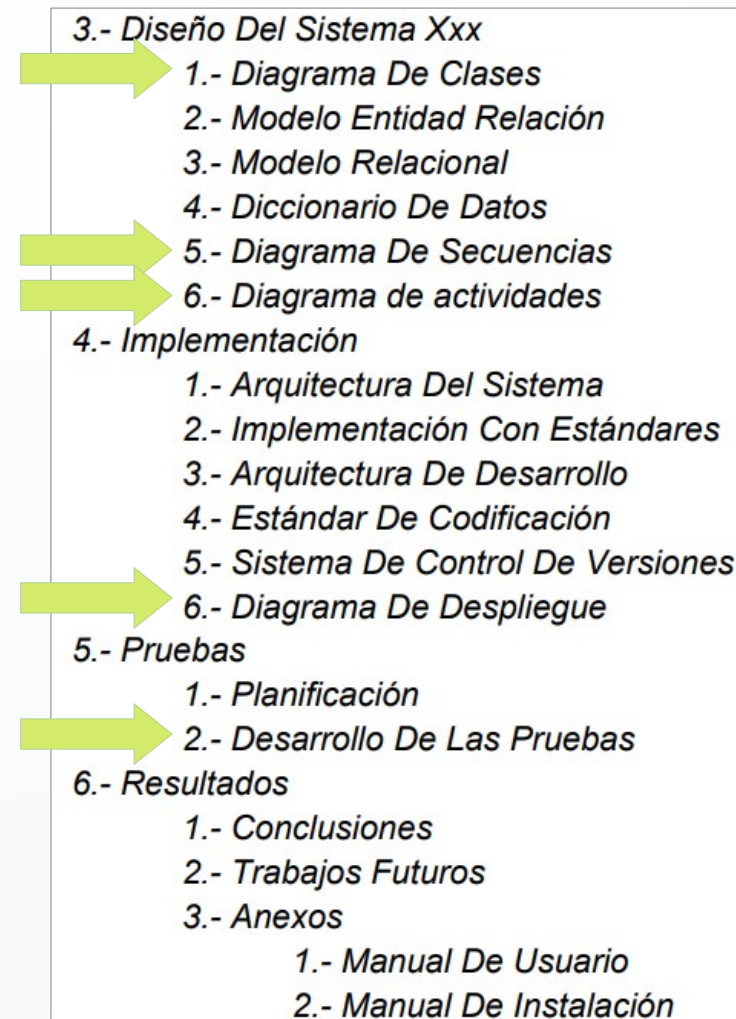
¡PREGUNTA CASI
SEGURA DEL EXAMEN!

5.2 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 05.DOCUMENTACIÓN

5 DOCUMENTACIÓN

5.1 ¿DOCUMENTAMOS?

5.2 DOCUMENTACIÓN EXTERNA

5.3 DOCUMENTACIÓN INTERNA

5.4 HERRAMIENTA JAVADOC

- PRÁCTICA NO EVALUABLE SOBRE JAVADOC



5.3 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.**



5.3 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

5.3 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.**

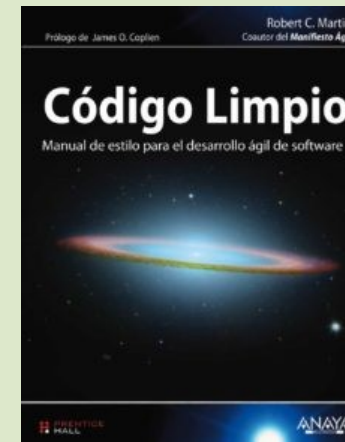


5.3 DOCUMENTACIÓN INTERNA

- Una vez que se...
debe resultar c...
 - En otras p...
- El có...
- El bu...
- Esto...
- Sin embargo, s...
- La posibilidad...
- aparece en tod...
- Los comentari...

Consejos para un código limpio:

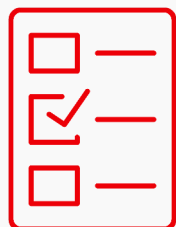
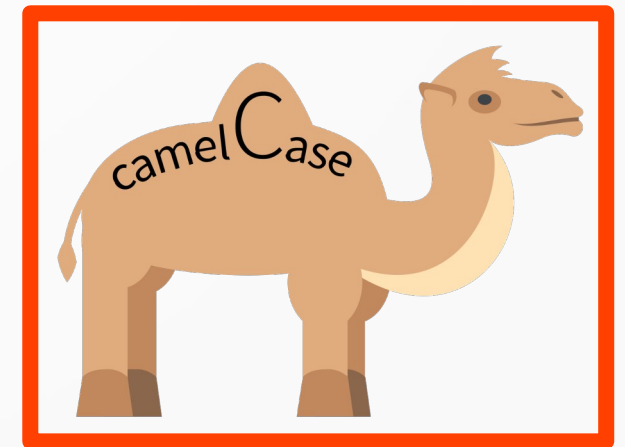
- 1) Evitar el uso de complicadas comparaciones condicionales
- 2) Eliminar las comparaciones con condiciones negativas
- 3) Evitar un gran anidamiento de bucles o de condiciones
- 4) Usar paréntesis para clarificar las expresiones lógicas o aritméticas
- 5) Usar espacios y/o símbolos claros para aumentar la legibilidad
- 6) **Pensar ”¿Podría yo entender esto si no fuera la persona que lo codificó?”**



ón y
miento

5.3 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**



¡PREGUNTA CASI
SEGURA DEL EXAMEN!

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

5.3 DOCUMENTACIÓN INTERNA

- **Comentario de prólogo**

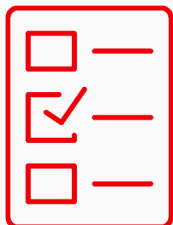
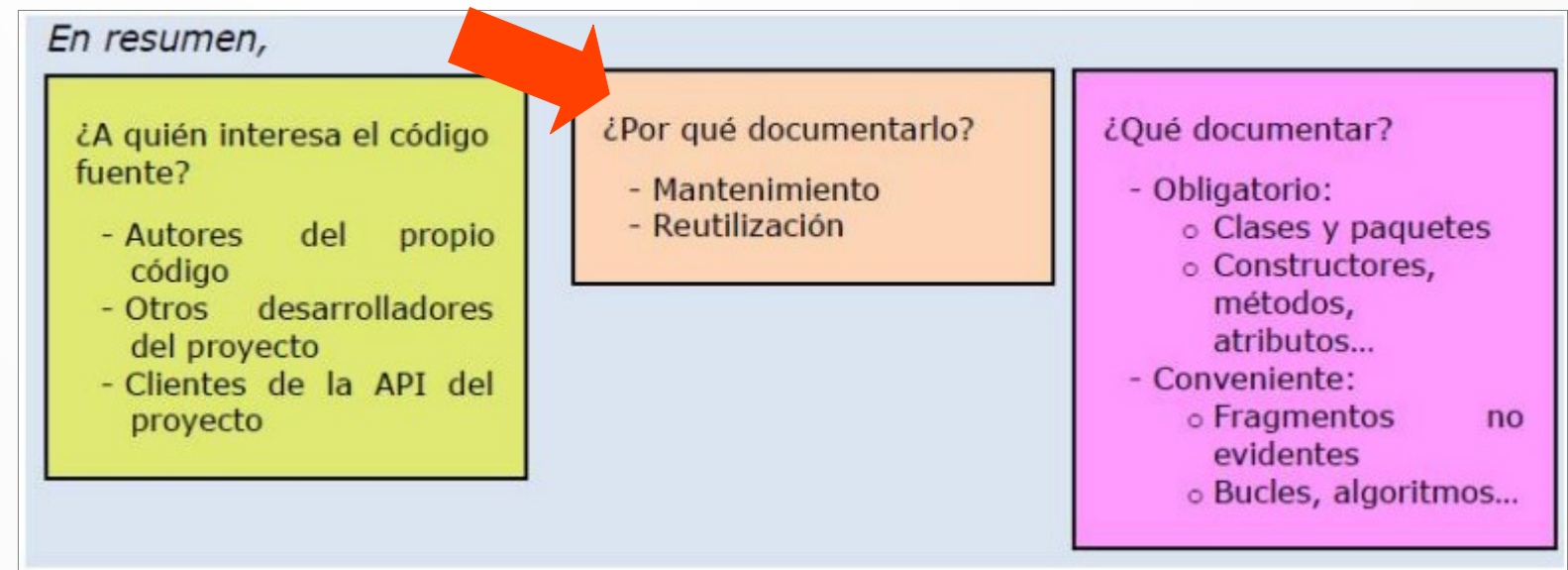
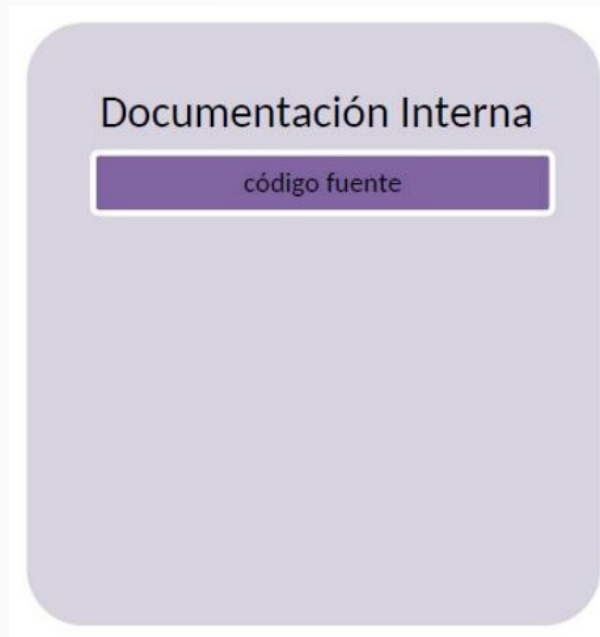
- Al principio de cada módulo debe haber un comentario de prólogo que indique el nombre de la aplicación/módulo/fichero, resuma su función y describa su autor.
- Se propone la siguiente estructura (igual de válida de muchas otras):
 1. Una sentencia que indique la función del módulo.
 2. Una descripción 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...

5.3 DOCUMENTACIÓN INTERNA



¡PREGUNTA CASI
SEGURA DEL EXAMEN!

NOTE: OF COURSE ANY ORGANIZATION CAN USE THE LIGHTWEIGHT FORM (E.G. BY CREATING SELF-SUFFICIENT TEAMS)

DOCUMENTATION

UX Knowledge Base Sketch #72

+ ANOTHER EXAMPLES: AGENCIES & FREELANCERS WHO'RE GETTING PAID FOR CREATING DOCUMENTATIONS. IF THEY WANT TO SWITCH TO LIGHTWEIGHT, FREQUENT CLIENT INVOLVEMENT CAN HELP.

LIGHTWEIGHT

E.G.: AGILE + LEAN PROCESSES (BUT IN CASE OF THIRD PARTY DEVELOPMENT, YOU CAN'T DO EVERYTHING THE LEAN WAY)

IT CAN ALSO BE A MIX, A TRANSITION: AT THE BEGINNING OF THE PROJECT: LEAN, LATER: STANDARDIZED DOCUMENTATION

E.G.: ENTERPRISE-INTERNAL & REGULATORY COMPLIANCE

HEAVY WEIGHT



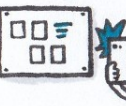
CLOSE COLLABORATION OF A CROSS-FUNCTIONAL TEAM EVERYONE IS ALREADY ON THE SAME PAGE.



"CUSTOM" CONTENT & STRUCTURE:
• WHAT IS THE MOST EFFECTIVE WAY?
• MINIMUM AMOUNT OF INFORMATION



IDEAL STAKEHOLDER MINDSET: FOCUSING ON THE OUTCOMES, IMPACT ACHIEVED BY THE PROJECT



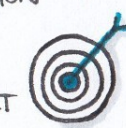
SOME FORMS: SKETCHES (PAPER OR WHITEBOARD), PROTOTYPES (EVEN PAPER PROTOTYPES - YOU CAN EVOLVE IT BASED ON TEST RESULTS), DOCUMENTING COLLABORATIVE SESSIONS (E.G. CELLPHONE PHOTOS)



• THE DOCUMENTATION CONSISTS OF LIVING DOCUMENTS - EDITED & UPDATED CONTINUALLY
• CO-CREATED DOCUMENTATION - FEEDBACK & QUESTIONS



IDEAL FOR ITERATIVE & INCREMENTAL PRODUCT DEVELOPMENT PROCESSES "WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION"



MAIN GOALS:
• ELIMINATING WASTE
• MOVING FORWARD FASTER
• CAPTURE CONVERSATIONS (→ RECALL)



THE RELEASED PRODUCT/FEATURE ITSELF IS ALSO A DOCUMENTATION
→ CONTINUOUS DELIVERY & IMPROVEMENT → HEAVY DOCUMENTATION WOULD QUICKLY BECOME OBSOLETE



IN MANY CASES: SILOS, ISOLATED TEAMS



STRICT STANDARDS & RULES BINDING CONTRACTS



USUAL STAKEHOLDER MINDSET: FOCUSING ON SHINY DESIGN ARTIFACTS & DELIVERABLES



SOME FORMS: REQUIREMENTS DOCUMENTS HUGE PILE OF DETAILED SPECIFICATION BIG DESIGN UP FRONT



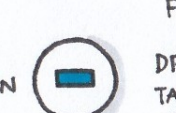
THE DOCUMENTATION IS "CARVED IN STONE", IT'S HARDER TO MODIFY IT LATER IN THE PROCESS



APPROPRIATE FOR WATERFALL PRODUCT DEVELOPMENT PROCESSES WITH BIG HAND-OFFS



MAIN GOALS, REASONS:
• THE PROCESS HEAVILY RELIES ON DOCUMENTATION
• IT NEEDS TO BE UNDERSTANDABLE FOR OTHER TEAMS



DRAWBACKS: TAKES HUGE AMOUNT OF TIME & CREATES A LOT OF WASTE (BUT SOMETIMES IT IS NECESSARY)

ADVICE



WHO IS THE TARGET AUDIENCE OF YOUR DOCUMENTATION? → DESIGN IT!

- FOR YOURSELF
- FOR YOUR TEAM
- FOR THE CLIENT
- FOR A THIRD PARTY TEAM

DIFFERENT GOALS
• MOTIVATION
• TERMINOLOGY

UX OF THE DOCUMENTATION, E.G. FINDABILITY

E.G. FOR DEVELOPERS:
• IMPLEMENTATION
• DATA MODELING
• DATA TYPES ETC.



YOUR DESIGN PROCESS SHOULD DETERMINE WHAT THINGS YOU DOCUMENT, DOCUMENTATION SHOULD COMPLEMENT & ENHANCE THE PROCESS (E.G.: RESEARCH, SYNTHESIZING, IDEATION, DESIGN, TEST)



ORGANIZE LIVE DEMO SESSIONS, INVITE YOUR TEAM TO OBSERVE USABILITY TESTS → FIRSTHAND INFORMATION IS THE BEST!

RESULT: NO DEPENDENCY ON THE DOCUMENTATION



→ SINGLE SOURCE OF TRUTH
DOCUMENTATION SHOULD BE EASILY ACCESSIBLE FOR THE WHOLE TEAM
• COLLABORATIVE: CONTRIBUTIONS, VERSIONING
• KEEP IT UP TO DATE



COMBINE WORDS & ILLUSTRATIONS (VISUALS) E.G. ANIMATED GIFS



CREATE A GRAVEYARD FOR DISCARDED VERSIONS, IDEAS



INCLUDE WHAT PROBLEMS YOU ARE ADDRESSING, ADD CONTEXT, DISCUSSIONS, INSPIRATIONS...

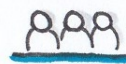
COMMON GOALS



• DOCUMENTING YOUR WAY OF THINKING
• INFORMATION FOR YOUR FUTURE SELF/ A TEAM/ THIRD PARTIES



• CREATING A DECISION-MAKING TOOL (MORE INFORMED DECISIONS)



• TEAM-ALIGNMENT, SHARED VISION



• EXPLAINING THE "HOW" AND THE "WHY"

(AND SOMETIMES THE GOAL IS TO BE IN ACCORDANCE WITH A CONTRACT)

UD 05.DOCUMENTACIÓN

5 DOCUMENTACIÓN

5.1 ¿DOCUMENTAMOS?

5.2 DOCUMENTACIÓN EXTERNA

5.3 DOCUMENTACIÓN INTERNA

5.4 HERRAMIENTA JAVADOC

- PRÁCTICA NO EVALUABLE SOBRE JAVADOC



5.4 HERRAMIENTA JAVADOC

- **HERRAMIENTA: JAVADOC**

- Javadoc es

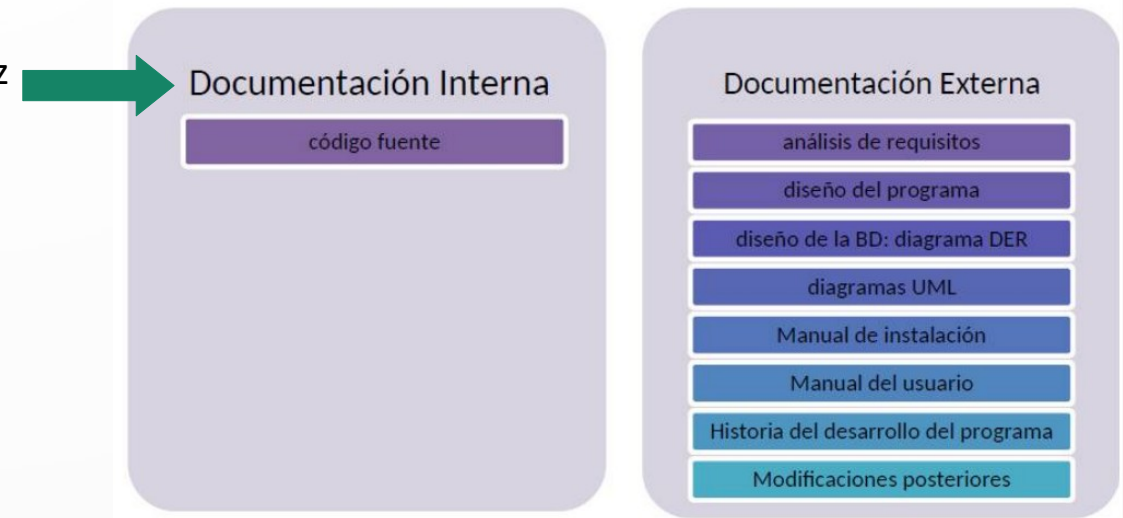
... herramienta/módulo/plugin/**aplicación/programa** para el lenguaje java, que permite generar la documentación de la interfaz o API (Application Programming Interface) de los **programas** desarrollados.

- **¿Qué hace?**

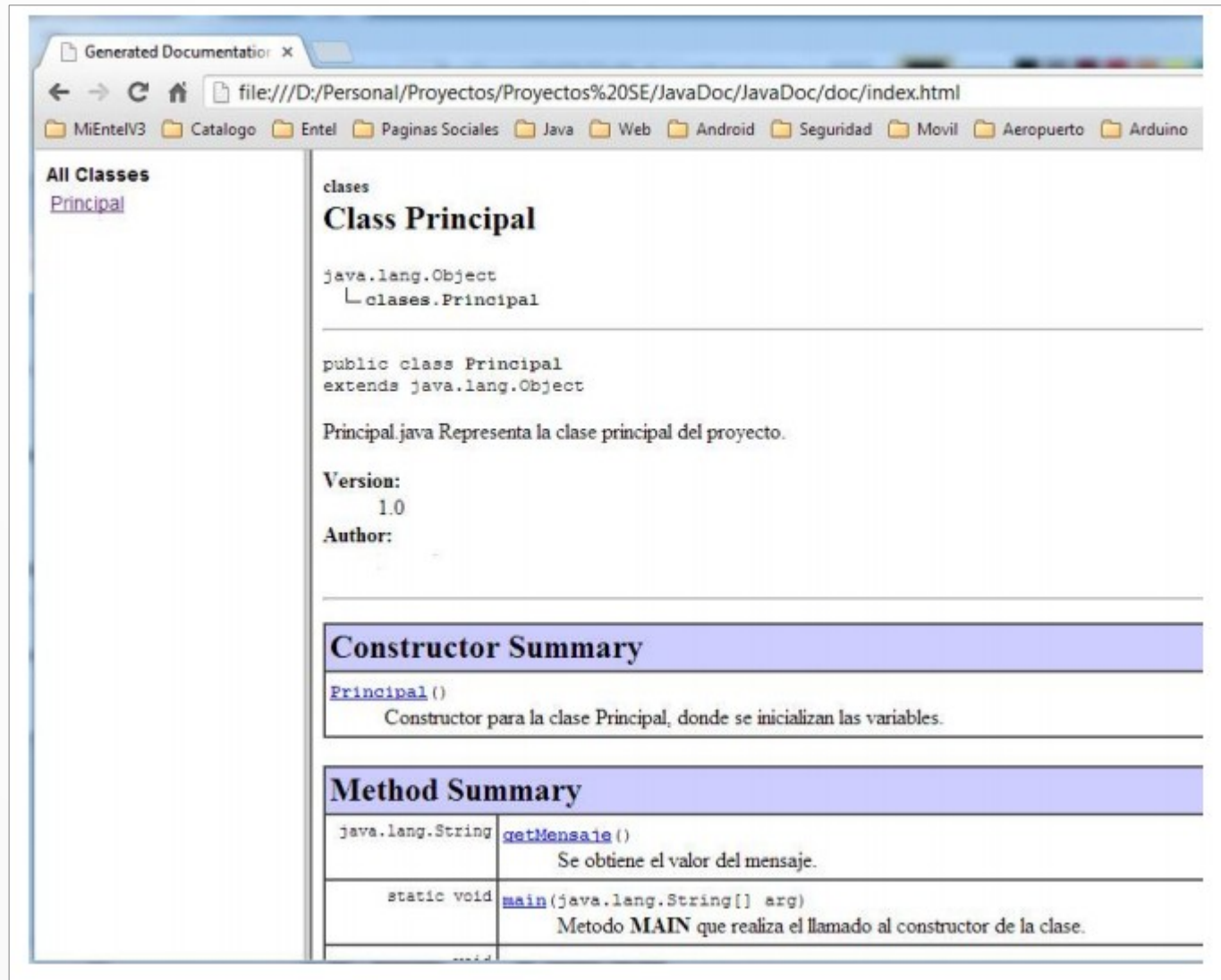
- Analiza las declaraciones y los comentarios del código fuente
 - Produce páginas html que ...
... describen clases, clases internas (subclases), interfaces, constructores y atributos de la clase (campos).

- **¿Para qué sirve?**

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



5.4 HERRAMIENTA JAVADOC



The screenshot displays a web browser window with the address bar showing the file path: `file:///D:/Personal/Proyectos/Proyectos%20SE/JavaDoc/JavaDoc/doc/index.html`. The browser's tab is labeled "Generated Documentation". The page content is organized into a sidebar and a main area.

Sidebar: Under the heading "All Classes", there is a link for [Principal](#).

Main Area:

- Under the heading "classes", it shows the class hierarchy: `java.lang.Object` is the superclass, and `classes.Principal` is the subclass.
- The class declaration is shown: `public class Principal extends java.lang.Object`.
- A description follows: "Principal.java Representa la clase principal del proyecto."
- Metadata is listed: "Version: 1.0" and "Author:".
- A section titled "Constructor Summary" contains the entry for `Principal()` with the description: "Constructor para la clase Principal, donde se inicializan las variables."
- A section titled "Method Summary" contains a table of methods:


Return Type	Method Name	Description
<code>java.lang.String</code>	<code>getMensaje()</code>	Se obtiene el valor del mensaje.
<code>static void</code>	<code>main(java.lang.String[] arg)</code>	Metodo MAIN que realiza el llamado al constructor de la clase.

5.4 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 )
 */
```

5.4 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\(Object\)](#)

5.4 HERRAMIENTA JAVADOC

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



5.4 HERRAMIENTA JAVADOC

- ¿Qué dice la Comunidad (de desarrolladores)?

stackoverflow.com/questions/1141228/javadoc-like-documentation-for-c

Javadoc-like Documentation for C++

Asked 11 years, 4 months ago · Active 1 year, 8 months ago · Viewed 37k times

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.

54

java c++ javadoc

12

share improve this question follow

edited Mar 8 '11 at 0:47

dreadwail 13.6k ● 20 ● 59 ● 90

asked Jul 17 '09 at 2:44

DHamrick 7,921 ● 9 ● 42 ● 58

add a comment

5 Answers

Active Oldest Votes

¿No encuentras la respuesta? [Pregunta en Stack Overflow en español.](#) ×

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).

Two important issues when choosing the documentation system is to make sure that it allows you to

- 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.
- Getting the output formatted as you want. It is preferable when the documentation fits in with your general project style.

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.

5.4 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.



5.4 HERRAMIENTA JAVADOC

Introducción a JavaDoc



Publicado con licencia reutilizable por enrique triviño

29 de noviembre de 2014



abrir el prezi