Resumen

[... versión del resumen en español ...]

En una página como máximo, el resumen explicará de modo breve la problemática que trata de resolver el trabajo *(el 'qué')*, la metodología para abordar su solución *(el 'cómo')* y los resultados obtenidos. En los trabajos cuyo idioma principal sea el inglés, el orden de Resumen y Abstract se invertirá.

Abstract

[... english version of the abstract ...]

English version for the abstract.

AGRADECIMIENTOS

Aunque es un apartado opcional, haremos bueno el refrán *«es de bien nacidos, ser agradecidos»* si empleamos este espacio como un medio para agradecer a todos los que, de un modo u otro, han hecho posible que el trabajo realizado *llegue a buen puerto*. Esta sección es ideal para agradecer a directores, profesores, mentores, familiares, compañeros, amigos, etc.

Estos agradecimientos pueden ser tan personales como se desee e incluir anécdotas y chascarrillos, pero *nunca deberían ocupar más de una página.*

Jesús Salido Tercero
Ciudad Real, 2021

NOTACIÓN Y ACRÓNIMOS

NOTACION

Ejemplo de lista con notación (o nomenclatura) empleada en la memoria del TFG.[1]

A, B, C, D	:	Variables lógicas
f, g, h	:	Funciones lógicas
$\cdot$	:	Producto lógico (AND), a menudo se omitirá como en AB en lugar de $A \cdot B$
$+$	:	Suma aritmética o lógica (OR) dependiendo del contexto
$\oplus$	:	OR exclusivo (XOR)
$\overline{A}$ o A'	:	Operador NOT o negación

LISTA DE ACRÓNIMOS

Ejemplo de lista con los acrónimos empleados en el texto.

CASE	:	Computer-Aided Software Engineering
CTAN	:	Comprenhensive TEX Archive network
IDE	:	Integrated Development Environment
ECTS	:	European Credit Transfer and Accumulation System
OOD	:	Object-Oriented Design
PhD	:	Philosophiae Doctor
RAD	:	Rapid Application Development
SDLC	:	Software Development Life Cycle
SSADM	:	Structured Systems Analysis & Design Method
TFE	:	Trabajo Fin de Estudios
TFG	:	Trabajo Fin de Grado
TFM	:	Trabajo Fin de Máster
UML	:	Unified Modeling Language

[1] Se incluye únicamente con propósito de ilustración, ya que el documento no emplea la notación aquí mostrada.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS

ÍNDICE DE ALGORITMOS

CAPÍTULO 1

INTRODUCCIÓN

Este capítulo aborda la motivación del trabajo. Se trata de señalar la necesidad que lo origina, su actualidad y pertinencia. Puede incluir también un estado de la cuestión (o estado del arte) en la que se revisen estudios o desarrollos previos y en qué medida sirven de base al trabajo que se presenta.

En este capítulo debería introducirse el *contexto disciplinar y tecnológico* en el que se desarrolla el trabajo de modo que pueda entenderse con facilidad el ámbito y alcance del TFG. Puesto que un TFG no tiene que ser necesariamente un trabajo con aportes novedosos u originales, solo es necesario la inclusión de «estado del arte» cuando este contribuya a aclarar aspectos clave del TFG o se desee justificar la originalidad del trabajo realizado.

A continuación se muestran algunos ejemplos para la inclusión de elementos en el documento como listas, tablas y figuras. Todos estos ejemplos se han presentado en al curso de LaTeX[6] y tanto estos como los recogidos en varias obras de referencia se pueden emplear para preparar este documento [1-4, 9].

IMPORTANTE: *A la hora de redactar el texto se debe poner especial atención en no cometer plagio y respetar los derechos de propiedad intelectual [5, 7]. En particular merece gran atención la inclusión de figuras e imágenes procedentes de Internet que no sean de elaboración propia. En este sentido se recomienda consultar el manual de la Universidad de Cantabria en el que se explica de modo conciso cómo incluir imágenes en un trabajo académico [8].*

1.1. EJEMPLOS DE LISTAS

A continuación se van a añadir algunos ejemplos que pueden emplearse al redactar la memoria.

Ejemplo de lista con *bullet* especial.

- ✔ peras
- manzanas
- naranjas

Ejemplo de lista en varias columnas.

1. peras
2. manzanas
3. naranjas
4. patatas
5. calabazas
6. fresas

Las listas se pueden personalizar (aunque debería estar justificado el cambiar el estilo por defecto, haciéndolo con mucha prudencia):

❶ peras

❷ manzanas

❸ naranjas

1.2. EJEMPLOS DE TABLAS

A continuación se incluyen algunos ejemplos de tablas hechas con LaTeX y paquetes dedicados.

Tabla 1.1: Ejemplo de uso de la macro `cline`

7C0	hexadecimal
3700	octal
11111000000	binario
1984	decimal

Ejemplo de tabla en la que se controla el ancho de la celda.

Tabla 1.2: Ejemplo de tabla con especificación de anchura de columna

Día	Temp Mín (°C)	Temp Máx (°C)	Previsión
Lunes	11	22	Día claro y muy soleado. Sin embargo, la brisa de la tarde puede hacer que las temperaturas desciendan
Martes	9	19	Nuboso con chubascos en muchas regiones. En Cataluña claro con posibilidad de bancos nubosos al norte de la región
Miércoles	10	21	La lluvia continuará por la mañana, pero las condiciones climáticas mejorarán considerablemente por la tarde

1.3. EJEMPLOS DE FIGURAS

En esta sección se añaden ejemplos de muestra para la inclusión de figuras simples y subfiguras.

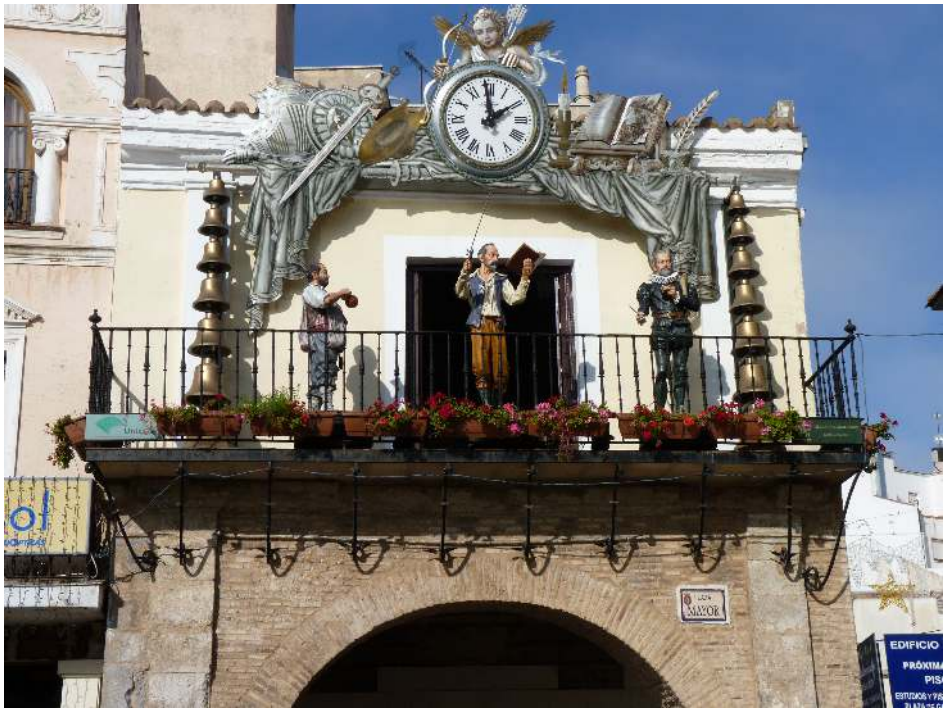
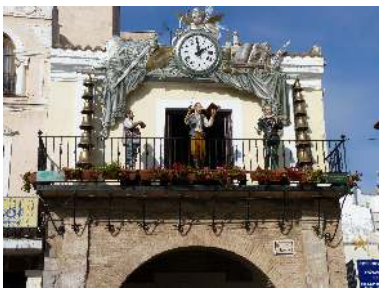
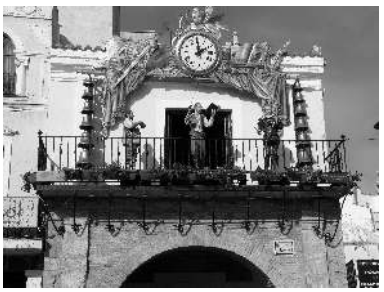


Figura 1.1: Fotografía a color (por J. Salido, CC BY-NC-ND)

Ejemplo de figuras compuestas por subfiguras incluidas con paquete `subcaption`.[1]



(a) Fotografía a color



(b) Fotografía en blanco y negro

Figura 1.2: Ejemplo de inclusión de subfiguras en un mismo entorno (por J. Salido, CC BY-NC-ND)

Mediante el uso de etiquetas (`\label`) es posible incluir referencias cruzadas a subfiguras como la fotografía en blanco y negro de la Fig. 1.2b.

En los trabajos académicos la inclusión de imágenes y figuras que no son propiedad del autor suscitan bastante controversia y son fuente de incumplimiento inadvertido de la ley de propiedad intelectual. Es importante recordar que *«el desconocimiento de la ley no exime de su cumplimiento»* por lo que se recomienda tanto a estudiantes como tutores consultar documentación informativa sobre el uso correcto de figuras en documentos académicos [8]. Entre las «incorrecciones» más frecuentes al incluir figuras en los documentos académicos se observan:

- Abuso del derecho de cita. Se produce al incluir, con fines exclusivamente decorativos o ilustrativos de la explicación, una figura sujeta a derechos de uso restringido invocando el derecho de cita (incluso con correcta atribución de la obra).

[1] https://osl.ugr.es/CTAN/macros/latex/contrib/caption/subcaption.pdftext

- Incorrecta atribución de la obra. Es habitual confundir al autor de la obra con la fuente de origen de la misma. La fuente es precisa cuando se cita la obra original. Sin embargo, la licencia de muchas obras exige la atribución al autor y la inclusión de la licencia bajo la que se distribuye o hace uso de la misma (véase como ejemplo cómo se realiza una correcta atribución en las Fig. 1.1 y 1.2 mencionando al autor y la licencia Creative-Commons[2] bajo la que se rige el uso de la imagen y el mecanismo de título alternativo para que dicha atribución no aparezca en el índice de figuras).
- Supresión de denominación de licencia de uso. Al incluir obras de terceros debemos tener presente los términos de distribución de la misma e incluirlos junto a la atribución de su legítimo autor.

La inclusión de material de *dominio público* o sin restricciones de uso hace innecesaria la atribución al autor pero puede incluirse una nota de agradecimiento.[3]

1.4. EJEMPLOS DE LISTADOS

Ejemplos más representativos de inclusión de porciones de código fuente.

Listado 1.1: Ejemplo de código fuente en lenguaje Java

```
1  // @author www.javadb.com
2  public class Main {
3  // Este método convierte un String a un vector de bytes
4
5  public void convertStringToByteArray() {
6
7  String stringToConvert = "This␣String␣is␣15";
8    byte[] theByteArray = stringToConvert.getBytes();
9    System.out.println(theByteArray.length);
10 }
11
12 public static void main(String[] args) {
13   new Main().convertStringToByteArray();
14 }
15 }
```

Otro ejemplo.

Listado 1.2: Ejemplo de código C

```
1 // Este código se ha incluido tal cual está en el fichero LaTeX
2 #include <stdio.h>
3
4 int main(int argc, char* argv[]) {
5   puts("¡Hola␣mundo!");
6 }
```

Ejemplo de entrada por consola.

```
$␣gcc␣-o␣Hola␣HolaMundo.c
```

Un ejemplo más. Este en Matlab:

[2] https://creativecommons.org
[3] Por cortesía de <x>.

Listado 1.3: Ejemplo escrito en Matlab

```
function f = fibonacci(n)
 % FIBONACCI Fibonacci sequence
 % f = FIBONACCI(n) generates the first n Fibonacci numbers.
 % Copyright 2014 Cleve Moler
 % Copyright 2014 The MathWorks, Inc.
f = zeros(n,1);
f(1) = 1;
f(2) = 2;
for k = 3:n
f(k) = f(k-1) + f(k-2);
end
```

1.4.1. Algoritmos con el paquete `algorithm2e`

Como ya se ha comentado en los textos científicos relacionados con las TIC[4] (Tecnologías de la Información y Comunicaciones) suelen aparecer porciones de código en los que se explica alguna función o característica relevante del trabajo que se expone. Muchas veces lo que se quiere ilustrar es un algoritmo o método en que se ha resuelto un problema abstrayéndose del lenguaje de programación concreto en que se realiza la implementación. El paquete `algorithm2e`[5] proporciona un entorno `algorithm` para la impresión apropiada de algoritmos tratándolos como objetos flotantes y con mucha flexibilidad de personalización. En el algoritmo 1.1 se muestra cómo puede emplearse dicho paquete. En este curso no se explican las posibilidades del paquete más en profundidad, ya que excede el propósito del curso. A todos los interesados se les remite a la documentación del mismo.

Algoritmo 1.1: Cómo escribir algoritmos

Datos : este texto
Resultado : como escribir algoritmos con LaTeX2e

```
inicialización;
while no es el fin del documento do
    leer actual;
    if comprendido then
        ir a la siguiente sección;
        la sección actual es esta;
    else
        ir al principio de la sección actual;
    end
end
```

1.5. MENÚS, PATHS Y TECLAS CON EL PAQUETE `menukeys`

Cada vez es más usual que los trabajos en ingeniería exijan el uso de software. Para poder especificar de modo elegante el uso menús, pulsación de teclas y directorios se recomienda el uso del paquete `menukeys`.[6] Este paquete nos permite especificar el acceso a un menú, por ejemplo:

Herramientas » Órdenes » PDFLaTeX

También un conjunto de teclas. Por ejemplo: Ctrl + ⇧ + T

[4] Por supuesto en un TFG (Trabajo Fin de Grado) o tesis de un centro superior de informática.
[5] https://osl.ugr.es/CTAN/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf
[6] https://osl.ugr.es/CTAN/macros/latex/contrib/menukeys/menukeys.pdf

O un directorio: C:▸user▸LaTeX▸Ejemplos

Aunque este paquete permite muchas opciones de configuración de los estilos aplicados, no es necesario hacerlo para obtener unos resultados muy elegantes.

1.6. EJEMPLOS DE FÓRMULAS MATEMÁTICAS

Para que LaTeX pueda incluir muchos símbolos matemáticos es preciso incluir algunos paquetes que ayudan en dicha tarea: `amsmath`, `amsfonts`, `amssymb`. También hay que tener en cuenta que si el tipo principal empleado en el texto es Times y se desea utilizar un tipo coherente en las fórmulas es conveniente emplear el paquete *mathptmx* en vez de *Times*. Pero en este caso es recomendable incluir siempre paquetes adicionales para suministrar las otras dos familias de fuentes escalables (p. ej. `helvet` para familia palo seco y `couriers` para `monoespaciada`). Si no se hace esta última inclusión pueden obtenerse errores de difícil diagnóstico.

1.6.1. Fórmulas creadas en línea y con entorno `equation`

Es muy sencillo incluir fórmulas matemáticas sencillas en el mismo texto en el que se escribe. Por ejemplo, $c^2 = a^2 + b^2$ que podría ser la ecuación representativa del teorema de Pitágoras.

Las fórmulas también se pueden separar del texto para que aparezcan destacadas, así:

$$c^2 = \int \left(a^2 + b^2\right) \cdot dx$$

Pero si se desea, las ecuaciones pueden ser numeradas de forma automática e incluso utilizar referencias cruzadas a ellas:

$$a^2 = b^2 + c^2 \tag{1.1}$$

No hay que preocuparse demasiado por la tipografía empleada en las fórmulas pues LaTeX hace por nosotros «casi» todo el trabajo.[7]

Los ejemplos que aquí se muestran son muy sencillos pero LaTeX proporciona entornos específicos más potentes. Para mostrar algo «más sofisticado» añado dos ejemplos más. La ec. 1.2 que es un poquito más compleja y la ec. 1.4 que está recuadrada.

$$I = \int_{-\infty}^{\infty} f(x)\, dx \tag{1.2}$$

Un ejemplo de alineación de ecuación mediante entorno `flalign`:[8]

$$f(x) = -1{,}25x^2 + 1{,}5x \tag{1.3}$$

En este caso la versión con estrella (`flalign*`) suprime la numeración de la ecuación.

[7] Los matemáticos son muy exquisitos y no se conforman con cualquier cosa, pero nosotros debemos ser mucho menos pretenciosos si queremos resultados rápidos.

[8] Otra forma de conseguir el alineamiento a la izquierda de las ecuaciones se consigue añadiendo `fleqn` como opción de la clase del documento.

$$\boxed{R = \frac{L}{2} \cdot \frac{(v_d + v_i)}{(v_d - v_i)}} \tag{1.4}$$

Algunos otros cuadros en ecuaciones son p. ej. $x + y = \boxed{\Omega}$ o incluso el que se muestra a continuación (ec. 1.5) y que abarca todo el ancho de la línea:[9]

$$\sqrt[n]{1 + x + x^2 + x^3 + \ldots} \tag{1.5}$$

1.6.2. Ecuaciones en varias líneas con entornos `eqnarray` y `align`

A continuación se muestra un ejemplo de ecuación muy larga dividida en varias líneas:

$$\begin{aligned}(1+x)^n = {} & \\ & 1 + nx + \frac{n\,(n-1)}{2!}x^2 + \\ & \frac{n\,(n-1)\,(n-2)}{3!}x^3 + \\ & \frac{n\,(n-1)\,(n-2)\,(n-3)}{4!}x^4 + \\ & \ldots\end{aligned}$$

También se puede escribir varias ecuaciones en líneas sucesivas alineadas por algún elemento como se hace en el siguiente ejemplo de uso del entorno `align`:

$$f(x) = \cos x \tag{1.6}$$
$$f'(x) = -\sin x \tag{1.7}$$
$$\int_0^x f(y)dy = \sin x$$

En este último ejemplo se observa también cómo es posible suprimir la numeración de una de las ecuaciones con el comando (`\nonumber`).

Para terminar, un ejemplo más del control del espaciado horizontal empleando el entorno `array`:

$$f(n) = \begin{cases} n/2 & \text{si } n \text{ es par} \\ -(n+1)/2 & \text{si } n \text{ es impar} \end{cases}$$

[9] Adaptado del manual *Documentation for fancybox.sty: Box tips and tricks for LaTeX* de Timothy Van Zandt (2010).

CAPÍTULO 2

OBJETIVO

Introduce y motiva la problemática (i.e. *¿cuál es el problema que se plantea y por qué es interesante su resolución?*)

Debe concretar y exponer detalladamente el problema a resolver, el entorno de trabajo, la situación y qué se pretende obtener. También puede contemplar las limitaciones y condicionantes a considerar para la resolución del problema (lenguaje de construcción, equipo físico, equipo lógico de base o de apoyo, etc.). Si se considera necesario, esta sección puede titularse *Objetivos del TFG e hipótesis de trabajo*. En este caso, se añadirán las hipótesis de trabajo que el alumno pretende demostrar con su TFG.

Una de las tareas más complicadas al proponer un TFG es plantear su Objetivo. La dificultad deriva de la falta de consenso respecto de lo que se entiende por *objetivo* en un trabajo de esta naturaleza. En primer lugar se debe distinguir entre dos tipos de objetivo:

1. La *finalidad específica* del TFG que se plantea para resolver una problemática concreta aplicando los métodos y herramientas adquiridos durante la formación académica. Por ejemplo, *«Desarrollo de una aplicación software para gestionar reservas hoteleras* on-line*»*.
2. El *propósito académico* que la realización de un TFG tiene en la formación de un graduado. Por ejemplo, la *adquisición de competencias específicas de la especialización* cursada.

En el ámbito de la memoria del TFG se tiene que definir el primer tipo de objetivo, mientras que el segundo tipo es el que se añade al elaborar la propuesta de un TFG presentada ante un comité para su aprobación. *Este segundo tipo de objetivo no debe incluirse en la memoria y en todo caso solo debe hacerse en la sección de conclusiones finales.*[1]

Un objetivo bien planteado debe estar determinado en términos del *«producto final»* esperado que resuelve un problema específico. Es por tanto un sustantivo que debería ser *concreto* y *medible*. El Objetivo planteado puede pertenecer una de las categorías que se indica a continuación:

- *Diseño y desarrollo de «artefactos»* (habitual en las ingenierías),
- *Estudio* que ofrece información novedosa sobre un tema (usual en las ramas de ciencias y humanidades), y
- *Validación de una hipótesis* de partida (propio de los trabajos científicos y menos habitual en el caso de los TFG).

Estas categorías no son excluyentes, de modo que es posible plantear un trabajo cuyo objetivo sea el diseño y desarrollo de un «artefacto» y este implique un estudio previo o la validación de alguna hipótesis para guiar el proceso. En este caso y cuando el objetivo sea lo suficientemente amplio puede ser conveniente su descomposición en elementos más simples hablando de *subobjetivos*. Por ejemplo, un programa informático puede descomponerse en módulos o requerir un estudio

[1] En lagunas titulaciones es obligatorio que la memoria explique las competencias específicas alcanzadas con la realización del trabajo.

previo para plantear un nuevo algoritmo que será preciso validar. La descomposición de un objetivo principal en subobjetivos u objetivos secundarios debería ser natural (no forzada), bien justificada y sólo pertinente en los trabajos de gran amplitud.

Junto con la definición del objetivo del trabajo se puede especificar los *requisitos* que debe satisfacer la solución aportada. Estos requisitos especifican *características* que debe poseer la solución y *restricciones* que acotan su alcance. En el caso de un trabajo cuyo objetivo es el desarrollo de un «artefacto» los requisitos pueden ser *funcionales* y *no funcionales*.

Al redactar el objetivo de un TFG se debe evitar confundir los medios con el fin. Así es habitual encontrarse con objetivos definidos en términos de las *acciones* (verbos) o *tareas* que será preciso realizar para llegar al verdadero objetivo. Sin embargo, a la hora de planificar el desarrollo del trabajo si es apropiado descomponer todo el trabajo en *hitos* y estos en *tareas* para facilitar dicha *planificación*.

La categoría del objetivo planteado justifica modificaciones en la organización genérica de la memoria del trabajo. Así en el caso de estudios y validación de hipótesis el apartado de resultados y conclusiones debería incluir los resultados de experimentación y los comentarios de cómo dichos resultados validan o refutan la hipótesis planteada.

CAPÍTULO 3

METODOLOGÍA

En este capítulo se debe detallar las metodologías empleadas para planificación y desarrollo del trabajo, así como explicar de modo claro y conciso cómo se han aplicado dichas metodologías.

A continuación se incluye una guía rápida que puede ser de gran utilidad en la elaboración de este capítulo.

3.1. GUÍA RÁPIDA DE LAS METODOLOGÍAS DE DESARROLLO DE SOFTWARE

3.1.1. Proceso de desarrollo de software

El **proceso de desarrollo de software** se denomina también **ciclo de vida del desarrollo del software** (*SDLC, Software Development Life-Cycle*) y cubre las siguientes actividades:

1. **Obtención y análisis de requisitos** (*requirements analysis*). Es la definición de la funcionalidad del software a desarrollar. Suele requerir entrevistas entre los ing. de software y el cliente para obtener el 'qué' y 'cómo'. Permite obtener una *especificación funcional* del software.
2. **Diseño** (*SW design*). Consiste en la definición de la arquitectura, los componentes, las interfaces y otras características del sistema o sus componentes.
3. **Implementación** (*SW construction and coding*). Es el proceso de codificación del software en un lenguaje de programación. Constituye la fase en que tiene lugar el desarrollo de software.
4. **Pruebas** (*testing and verification*). Verificación del correcto funcionamiento del software para detectar fallos lo antes posible. Persigue la obtención de software de calidad. Consisten en pruebas de *caja negra* y *caja blanca*. Las primeras comprueban que la funcionalidad es la esperada y para ello se verifica que ante un conjunto amplio de entradas, la salida es correcta. Con las segundas se comprueba la robustez del código sometiéndolo a pruebas cuya finalidad es provocar fallos de software. Esta fase también incorpora la *pruebas de integración* en las que se verifica la interoperabilidad del sistema con otros existentes.
5. **Documentación** (*documentation*). Persigue facilitar la mejora continua del software y su mantenimiento.
6. **Despliegue** (*deployment*). Consiste en la instalación del software en un entorno de producción y puesta en marcha para explotación. En ocasiones implica una fase de *entrenamiento* de los usuarios del software.
7. **Mantenimiento** (*maintenance*). Su propósito es la resolución de problemas, mejora y adaptación del software en explotación.

3.1.2. Metodologías de desarrollo software

Las metodologías son el modo en que las fases del proceso software se organizan e interaccionan para conseguir que dicho proceso sea reproducible y predecible para aumentar la productividad y la calidad

del software.

Una metodología es una colección de:

- **Procedimientos** (indican cómo hacer cada tarea y en qué momento),
- **Herramientas** (ayudas para la realización de cada tarea), y
- **Ayudas documentales**.

Cada metodología es apropiada para un tipo de proyecto dependiendo de sus características técnicas, organizativas y del equipo de trabajo. En los entornos empresariales es obligado, a veces, el uso de una metodología concreta (p. ej. para participar en concursos públicos). El estándar internacional ISO/IEC 12270 describe el método para seleccionar, implementar y monitorear el ciclo de vida del software.

Mientras que unas intentan sistematizar y formalizar las tareas de diseño, otras aplican técnicas de gestión de proyectos para dicha tarea. Las metodologías de desarrollo se pueden agrupar dentro de varios enfoques según se señala a continuación.

1. **Metodología de Análisis y Diseño de Sistemas Estructurados** (*SSADM, Structured Systems Analysis and Design Methodology*). Es uno de los paradigmas más antiguos. En esta metodología se emplea un modelo de desarrollo en cascada (*waterfall*). Las fases de desarrollo tienen lugar de modo secuencial. Una fase comienza cuando termina la anterior. Es un método clásico poco flexible y adaptable a cambios en los requisitos. Hace especial hincapié en la planificación derivada de una exhaustiva definición y análisis de los requisitos. Son metodologías que no lidian bien con la flexibilidad requerida en los proyectos de desarrollo software. Derivan de los procesos en ingeniería tradicionales y están enfocadas a la reducción del riesgo. Emplea tres técnicas clave:
   - Modelado lógico de datos (*Logical Data Modelling*),
   - Modelado de flujo de datos (*Data Flow Modelling*), y
   - Modelado de Entidades y Eventos (*Entity Event Modelling*).
2. **Metodología de Diseño Orientado a Objetos** (*OOD, Object-Oriented Design*). Está muy ligado a la OOP (Programación Orientada a Objetos) en que se persigue la reutilización. A diferencia del anterior, en este paradigma los datos y los procesos se combinan en una única entidad denominada *objetos* (o clases). Esta orientación pretende que los sistemas sean más modulares para mejorar la eficiencia, calidad del análisis y el diseño. Emplea extensivamente el Lenguaje Unificado de Modelado (UML) para especificar, visualizar, construir y documentar los artefactos de los sistemas software y también el modelo de negocio. UML proporciona una serie diagramas de básicos para modelar un sistema:
   - Diagrama de Clase (*Class Diagram*). Muestra los objetos del sistema y sus relaciones.
   - Diagrama de Caso de Uso (*Use Case Diagram*). Plasma la funcionalidad del sistema y quién interacciona con él.
   - Diagrama de secuencia (*Sequence Diagram*). Muestra los eventos que se producen en el sistema y como este reacciona ante ellos.
   - Modelo de Datos (*Data Model*).
3. **Desarrollo Rápido de Aplicaciones** (*RAD, Rapid Application Developmnent*). Su filosofía es sacrificar calidad a cambio de poner en producción el sistema rápidamente con la funcionalidad esencial. Los procesos de especificación, diseño e implementación son simultáneos. No se realiza una especificación detallada y se reduce la documentación de diseño. El sistema se diseña en una serie de pasos, los usuarios evalúan cada etapa en la que proponen cambios y nuevas mejoras. Las interfaces de usuario se desarrollan habitualmente mediante sistemas interactivos de desarrollo. En vez de seguir un modelo de desarrollo en cascada sigue un modelo en espiral (Boehm). La clave de este modelo es el desarrollo continuo que ayuda a minimizar los riesgos. Los desarrolladores deben definir las características de mayor prioridad. Este tipo de desarrollo se basa en la creación de prototipos y realimentación obtenida de los clientes para

definir e implementar más características hasta alcanzar un sistema aceptable para despliegue.

4. **Metodologías Ágiles**. *"[...] envuelven un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Así el trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo. Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Teniendo gran importancia el concepto de "Finalizado"(Done), ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar el lanzamiento del producto al mercado, sino incrementar el valor por medio de "software que funciona"(sin errores). Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. [...]"*[1]

3.1.3. Proceso de testing

1. *Pruebas modulares* (pruebas unitarias). Su propósito es hacer pruebas sobre un módulo tan pronto como sea posible. Las *pruebas unitarias* que comprueban el correcto funcionamiento de una unidad de código. Dicha unidad elemental de código consistiría en cada función o procedimiento, en el caso de programación estructurada y cada clase, para la programación orientada a objetos. Las características de una prueba unitaria de calidad son: *automatizable* (sin intervención manual), *completa*, *reutilizable*, *independiente* y *profesional*.
2. *Pruebas de integración*. Pruebas de varios módulos en conjunto para comprobar su interoperabilidad.
3. *Pruebas de caja negra*.
4. *Beta testing*.
5. *Pruebas de sistema y aceptación*.
6. *Training*.

3.1.4. Herramientas CASE (*Computer Aided Software Engineering*)

Las herramientas CASE están destinadas a facilitar una o varias de las tareas implicadas en el ciclo de vida del desarrollo de software. Se pueden dividir en la siguientes categorías:

1. Modelado y análisis de negocio.
2. Desarrollo. Facilitan las fases de diseño y construcción.
3. Verificación y validación.
4. Gestión de configuraciones.
5. Métricas y medidas.
6. Gestión de proyecto. Gestión de planes, asignación de tareas, planificación, etc.

IDE (Integrated Development Environment)

- Notepad++
- Visual Studio Code
- Atom
- GNU Emacs
- NetBeans
- Eclipse
- Qt Creator
- jEdit
- ItelliJ IDEA

Depuración

- GNU Debugger

Testing

[1] Fuente: Wikipedia

- JUnit. Entorno de pruebas para Java.
- CUnit. Entorno de pruebas para C.
- PyUnit. Entorno de pruebas para Python.

Repositorios y control de versiones

- Git
- Mercurial
- Github
- Bitbucket
- SourceTree

Documentación

- LaTeX
- Markdown
- Doxygen
- DocGen
- Pandoc

Gestión y planificación de proyectos

- Trello
- Jira
- Asana
- Slack
- Basecamp
- Teamwork Projects
- Zoho Projects

3.1.5. Fuentes de información adicional

- Top 6 Software Development Methodologies. Maja Majewski. Planview LeanKit, 2019.
- 12 Best software development methodologies with pros and cons. acodez, 2018.
- Software Development Methodologies. Association of Modern Technologies Professionals, 2019.

CAPÍTULO 4

RESULTADOS

En esta sección se describirá la aplicación del método de trabajo presentado en el capítulo 3, mostrando los elementos (modelos, diagramas, especificaciones, etc.) más importantes.

Este apartado debe explicar cómo el empleo de la metodología permite satisfacer tanto el objetivo principal como los específicos planteados en el TFG así como los requisitos exigidos (según exposición en cap. 2).

CAPÍTULO 5

CONCLUSIONES

En este capítulo se realizará un juicio crítico y discusión sobre los resultados obtenidos. *Cuidado, esta discusión no debe confundirse con una valoración del enriquecimiento personal que supone la realización del trabajo como culminación de una etapa académica.* Aunque de gran importancia, esta última valoración debe quedar fuera de la memoria del trabajo y solo debe ahondarse en ella ante requerimiento explícito del comité en el acto de defensa.

Si es pertinente deberá incluir información sobre trabajos derivados como publicaciones o ponencias en preparación, así como trabajos futuros *(solo si estos están iniciados o planificados en el momento que se redacta el texto)*. Evitar hacer una lista de posibles mejoras. Contrariamente a lo que alguno pueda pensar generalmente aportan impresión de trabajo incompleto o inacabado.[1]

5.1. JUSTIFICACIÓN DE COMPETENCIAS ADQUIRIDAS

Es muy importante recordar que según la normativa vigente, el capítulo de conclusiones debe incluir *obligatoriamente* un apartado destinado a justificar la aplicación en el TFG de competencias específicas (dos o más) asociadas a la tecnología específica cursada.

En el TFG se han trabajado las competencias correspondientes a la Tecnología Específica de *[poner lo que corresponda]*:

Código de la competencia 1: *[Texto de la competencia 1]*. Explicación de cómo dicha competencia se ha trabajado en el TFG.

Código de la competencia 2: *[Texto de la competencia 2]*. Explicación de cómo dicha competencia se ha trabajado en el TFG.

…otras más si las hubiera.

5.2. PLANIFICACIÓN Y COSTES

En este capítulo se puede incluir una valoración del trabajo realizado en el que se justifique el tiempo dedicado al TFG teniendo en cuenta que este tiene asignados 12 créditos ECTS que se traducen en 300-360 horas totales. En este sentido una correcta planificación del TFG debería garantizar que el trabajo se realice dentro de la horquilla señalada. Queda a criterio del tribunal la evaluación de valores extremos por defecto o exceso en función de los resultados obtenidos.

Es muy importante que todas las justificaciones aportadas se sustenten no solo en juicios de valor sino en evidencias tangibles como: historiales de actividad, repositorios de código y documentación, porciones de código, trazas de ejecución, capturas de pantalla, demos, etc.

[1] Puede reflexionarse en ello por si en la defensa del trabajo se pregunta sobre estas posibles mejoras.

BIBLIOGRAFÍA

[1] B. Cascales y P. Lucas, *El Libro de LaTeX*. Pearson Education, 2005, ISBN: 9788420537795.

[2] M. Goossens, S. Rahtz, and F. Mittelbach, *The LaTeX graphics companion*, 2nd ed. Addison-Wesley Reading, MA, 2007.

[3] L. Lamport, *LaTeX: A document preparation system*, 2nd ed. Addison-Wesley, 1994.

[4] T. Oetiker y col., *La introducción no-tan-corta a LaTeX2e*, ver. 5.03, 2014. dirección: http://www.ctan.org/tex-archive/info/lshort/spanish/.

[5] C. Regan, J. Haas y P. Stevens. «El plagio y la honestidad académica». T. U. of Sidney Library, ed. (2000), dirección: http://www.crue.org/tutorial_plagio/ (visitado 15-07-2019).

[6] J. Salido. «Curso: LaTeX esencial para preparación de TFG, Tesis y otros documentos académicos», Universidad de Castilla-La Mancha. (2010), dirección: http://visilab.etsii.uclm.es/?page_id=1468 (visitado 12-02-2017).

[7] The University of Sydney Library. «Cómo citar y elaborar referencias». The University of Sydney Library, ed. (2000), dirección: http://www.crue.org/tutorial_referencias/ (visitado 15-07-2019).

[8] Universidad de Cantabria. «Cómo usar imágenes en trabajos». U. de Cantabria, ed. (2018), dirección: https://web.unican.es/buc/Documents/Formacion/guia_imagenes.pdf (visitado 15-07-2019).

[9] WikiMedia. «LaTeX Wikibook». (2010), dirección: http://en.wikibooks.org/wiki/LaTeX (visitado 02-02-2017).

ANEXOS

ANEXO A

EL PRIMER ANEXO

Los anexos se incluirá de modo opcional material suplementario que podrá consistir en breves manuales, listados de código fuente, esquemas, planos, etc. Se recomienda que no sean excesivamente voluminosos, aunque su extensión no estará sometida a regulación por afectar esta únicamente al texto principal.

Bibliografía Esta sección, que si se prefiere puede titularse «Referencias», incluirá un listado por orden alfabético (primer apellido del primer autor) con todas las obras en que se ha basado para la realización del TFG en las que se especificará: autor/es, título, editorial y año de publicación. Solo se incluirán en esta sección las referencias bibliográficas que hayan sido citadas en el documento. Todas las fuentes consultadas no citadas en el documento deberían incluirse en una sección opcional denominada «Material de consulta», aunque preferiblemente estas deberían incluirse como referencias en notas a pie de página a lo largo del documento.

Se usará método de citación numérico con el número de la referencia empleada entre corchetes. La cita podrá incluir el número de página concreto de la referencia que desea citarse. Debe tenerse en cuenta que el uso correcto de la citación implica que debe quedar claro para el lector cuál es el texto, material o idea citado. Las obras referenciadas sin mención explícita o implícita al material concreto citado deberían considerarse material de consulta y por tanto ser agrupados como «Material de consulta» distinguiéndolas claramente de aquellas otras en las que si se recurre a la citación.

Cuando se desee incluir referencias a páginas genéricas de la Web sin mención expresa a un artículo con título y autor definido, dichas referencias podrán hacerse como notas al pie de página o como un apartado dedicado a las «Direcciones de Internet».

Todo el material ajeno deberá ser citado convenientemente sin contravenir los términos de las licencias de uso y distribución de dicho material. Esto se extiende al uso de diagramas y fotografías. El incumplimiento de la legislación vigente en materia de protección de la propiedad intelectual es responsabilidad exclusiva del autor independientemente de la cesión de derechos que este haya convenido.

Índice Temático Este índice es opcional y se empleará como índice para encontrar los temas tratados en el trabajo. Se organizará de modo alfabético indicando el número de página(s) en el que se aborda el tema concreto señalado.