UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

<Tecnología Específica>

TRABAJO FIN DE GRADO

Plantilla de TFG para ESI-UCLM
Curso de LaTeX esencial

Jesús Salido Tercero

<mes, año>

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

<Primera línea Depto. Director>
<Segunda línea Depto. Director>

<Tecnología Específica>

TRABAJO FIN DE GRADO

Plantilla de TFG para ESI-UCLM
Curso de LaTeX esencial

Autor: Jesús Salido Tercero

Tutor: <tutor(a) (nombre apellidos)>

Co-tutor: <co-tutor(a) (nombre apellidos)>

<mes, año>

Plantilla TFG

Tribunal:

Presidente: _____

Vocal: _____

Secretario: _____

Fecha de defensa: _____

Calificación: _____

Presidente	Vocal	Secretario
Fdo.:	Fdo.:	Fdo.:

A mis alumnos ...
(para siempre)

Resumen

(... versión del resumen en español ...)

El resumen debe ocupar como máximo una página y en dicho espacio proporcionará información crucial sobre el *'qué'* (problemática que trata de resolver el TFG), el *'cómo'* (metodología para llegar a los resultados) y los objetivos alcanzados.

Abstract

(... english version of the abstract ...)

Versión del resumen en inglés. En los trabajos cuyo idioma principal sea el inglés, el orden de Resumen y Abstract se invertirá.

AGRADECIMIENTOS

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

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 de la que surge, su actualidad y pertinencia. Puede incluir también un estado de la cuestión en la que se revisen estudios o desarrollos previos y en qué medida sirven de base al trabajo que se presenta.

A continuación se muestran algunos ejemplos para la inclusión de elementos en el documento.

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 compacta (también se puede emplear el entorno para enumeraciones *compactenum*)

- peras
- manzanas
- naranjas

Ejemplo de lista en varias columnas.

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

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

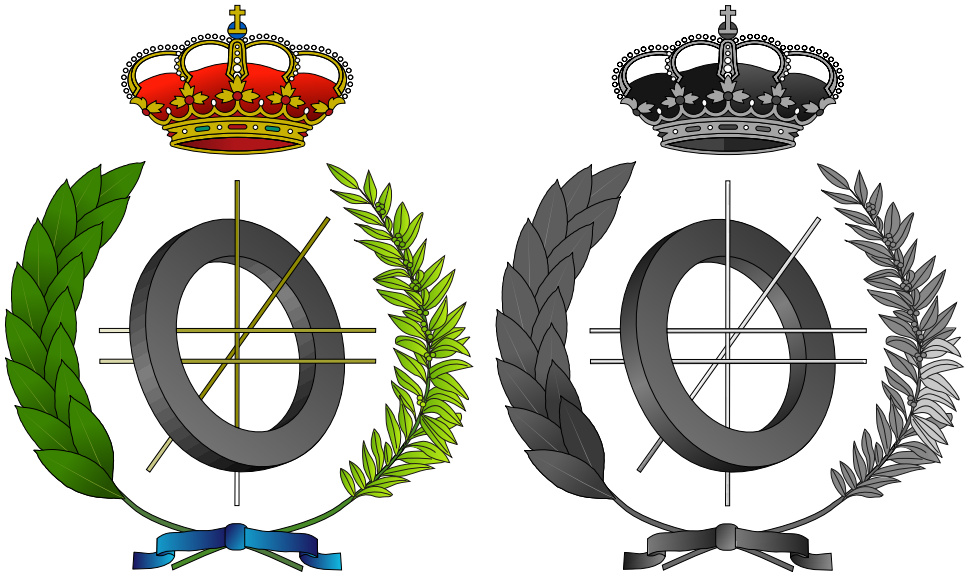
Día	Temp Mín (°C)	Temp Máx (°C)	Previsión
Lunes	11	22	Día claro y muy soleado. Sin embargo, la brisa de la tarde puede hacer que las temperaturas desciendan
Martes	9	19	Nuboso con chubascos en muchas regiones. En Cataluña claro con posibilidad de bancos nubosos al norte de la región
Miércoles	10	21	La lluvía 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: Figura vectorial del escudo de la ESI

Ejemplo de figuras compuestas por subfiguras. ¡Ojo! porque el paquete subfigure ha quedado obsoleto.



(a) Gráfico vectorial PDF (b) Gráfico png

Figura 1.2: Ejemplo de inclusión de subfiguras en un mismo entorno

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

```java
// @author www.javadb.com
public class Main {
// Este método convierte un String a
// un vector de bytes

public void convertStringToByteArray() {

String stringToConvert = "This␣String␣is␣15";
byte[] theByteArray = stringToConvert.getBytes();
System.out.println(theByteArray.length);
}

// argumentos de línea de comandos
public static void main(String[] args) {
new Main().convertStringToByteArray();
}
}
```

Otro ejemplo.

Listado 1.2: Ejemplo de código C

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

Ejemplo de entrada por consola.

```
$␣gcc␣-o␣Hola␣HolaMundo.c
```

1.4.1. Algoritmos con el paquete `algorithm2e`

Como ya se ha comentado en los textos científicos relacionados con las TIC[1] (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`[2] proporciona un entorno `algorithm` para la impresión apropiada de algoritmos tratándolos como objetos flotantes y con muchas 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.

[1] Por supuesto en un TFG o tesis de una Escuela de Informática.

[2] https://osl.ugr.es/CTAN/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf

Algoritmo 1.1: Cómo escribir algoritmos

Datos : este texto
Resultado : como escribir algoritmos con LaTeX2e

```
1  inicialización;
2  while no es el fin del documento do
3  |  leer actual;
4  |  if comprendido then
5  |  |  ir a la siguiente sección;
6  |  |  la sección actual es esta;
7  |  else
8  |  |  ir al principio de la sección actual;
9  |  end
10 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`.[3] Este paquete nos permite especificar el acceso a un menú, por ejemplo:

Herramientas ▸ Órdenes ▸ PDFLaTeX

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

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.

[3] https://osl.ugr.es/CTAN/macros/latex/contrib/menukeys/menukeys.pdf

CAPÍTULO 2

OBJETIVO

Introduce y motiva la problemática (i.e. *¿cuál es el problema que se plantea y porqué 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* de 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 de objetivo 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 el apartado correspondiente de la memoria y en todo caso puede valorarse su satisfacción en la sección de resultados y conclusiones.

Un objetivo bien planteado para el TFG 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 éste 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 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 TFG de gran amplitud.

Junto con la definición del objetivo del TFG 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 TFG 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 TFG. 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, componentes, 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 unos intentan sistematizar y formalizar las tareas de diseño, otros 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, mejorando la eficiencia y 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 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*). En el se 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 éste 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) De este modo se intenta 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.

[1] Wikipedia

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

- 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 en este caso concreto, mostrando los elementos (modelos, diagramas, especificaciones, etc.) más importantes. Este apartado debe explicar cómo la metodología satisface los objetivos y requisitos planteados.

CAPÍTULO 5

CONCLUSIONES

En este capítulo se realizará un juicio crítico y discusión sobre los resultados obtenidos. Si es pertinente deberá incluir información sobre trabajos derivados como publicaciones o ponencias, así como trabajos futuros, solo si estos están planificados en el momento en que se redacta el texto. Además incluirá obligatoriamente la explicación de cómo el trabajo realizado satisface las competencias de la tecnología específica cursada.

ANEXO A

EL PRIMER ANEXO

En 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 del trabajo independientemente de la cesión de derechos que este haya convenido. De este modo será responsable legal ante cualquier acción judicial derivada del incumplimiento de los preceptos aplicables. Así mismo ante dicha circunstancia los órganos académicos se reservan el derecho a imponer al autor la sanción administrativa que se estime pertinente.

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.

BIBLIOGRAFÍA

[1] Tobias Oetiker et al. *La introducción no-tan-corta a LaTeX2e*. Ver. 5.03. 2014. URL: http://www.ctan.org/tex-archive/info/lshort/spanish/.

[2] Alexánder Borbón y Walter Mora. *Edición de textos científicos con LaTeX. Composición, gráficos, diseño editorial y presentaciones beamer*. Ed. por Educación e Internet Revista Digital Matemática. Instituto Tecnológico de Costa Rica, 2017. ISBN: 978-9977-66-227-5. URL: https://tecdigital.tec.ac.cr/revistamatematica/Libros/LATEX/LaTeX_2013.pdf.

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

[4] B. Cascales y col. *LaTeX. Una imprenta en sus manos*. Aula Documental de Investigación, 2000.

[5] M. Goossens, F. Mittelbach y A. Samarin. *The LaTeX companion*. 2.ª ed. Addison-Wesley Reading, MA, 2004.

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

[7] H. Kopka y P.W. Daly. *A guide to LaTeX*. 4.ª ed. Addison-Wesley, 2004.

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

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

[10] WikiMedia. *LaTeX Wikibook*. 2010. URL: http://en.wikibooks.org/wiki/LaTeX (visitado 02-02-2017).

[11] Leonor Zozaya. *Redacción de textos. Recomendaciones para presentar trabajos académicos*. 2012-2017. URL: http://redaccion.hypotheses.org/ (visitado 21-06-2019).

ÍNDICE TEMÁTICO