

UNIVERSIDAD DE CASTILLA-LA MANCHA

BACHELOR IN COMPUTING ENGINEERING

Coursed intensification: Computer Engineering

BACHELOR DISSERTATION

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management

Miguel Ángel Ruiz Arreaza

Escuela
Superior
de Informática



Ciudad Real, Julio de 2025



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

Tutor's Department:

Coursed intensification: Computer Engineering

BACHELOR DISSERTATION

**MADTrack: Distributed System for Dataset and Artificial
Intelligence Model Management**

Author: Miguel Ángel Ruiz Arreaza

Tutor: José Luis Espinosa Aranda

Co-tutor: Pablo Tomás Toledano González

July, 2025

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management
© Miguel Ángel Ruiz Arreaza, 2025

The author may choose the license type they wish. This document is distributed under license CC BY-NC-SA 4.0. The full text of the license is available at <https://creativecommons.org/licenses/by-nc-sa/4.0/>. The copy and distribution of this work is permitted in all parts of the world, without royalties and in any medium, as long as this notice is preserved. In addition, permission is granted to copy and distribute translations of this book from the English original to another language, provided that the copyright notice and this permission notice are preserved in all copies.



This text has been prepared with the L^AT_EX template for Final Dissertations in Computing Engineering for UCLM, published by [Jesús Salido](#) in the public Zenodo repository, DOI: [10.5281/zenodo.4561708](https://doi.org/10.5281/zenodo.4561708).

*To my family, close friends, fellow colleagues, classmates and professors
For their patience and for accompanying me and showing me the way through this exciting journey.*

MADTrack: DS for Dataset and AI Model Management

Miguel Ángel Ruiz Arreaza

Ciudad Real, Julio 2025

Abstract

«What»

Artificial Intelligence is on the rise, and many companies are striving to create a wide-ranging variety of models to help themselves and their customers in an equally wide range of specific tasks. This inevitably translates into multiple model trainings being performed a year and, in many cases, many datasets being created or modified in the same time interval. Such is the case of UBOTICA Technologies: a Space:AI company that puts faith in Computation *on the edge* to deliver AI solutions integrated in embedded systems incorporated in spacial modules, which come with limited space for storing data and computing power. The development of these solutions requires multiple training and deployment iterations, which over the years led to the current tangled mess of untraceable datasets and models, which makes it difficult to search for a specific AI training configuration.

«How»

MADTrack is a distributed configuration management system with the purpose of putting order to the configurational chaos previously mentioned. It will store, track and manage all changes within datasets and AI model configurations. The main restrictions over the development of the system are the limited storage space of the company for this resources (the management of the evolution of datasets and models has to be done efficiently), the distributed nature of the environment where the items are stored and managed and the need for the system to be integrated in a greater processing workflow. The development of the system will be divided in a series of prototypes with an iterative and incremental approach, following continuous testing policies.

«Conclusion»

The resulting system will consist on a distributed system following the client-server application, with a local or remote server attending requests from multiple clients sending requests by means of a software wrapper library which in turn will interact with other open-source technologies.

Keywords: Escuela Superior de Informática, UCLM, TFG, L^AT_EX, T_EXLive, AI, Deep Learning, Distributed System, Dataset, configuration management .

MADTrack: Distributed System for Dataset and Artificial Intelligence Model Management

Miguel Ángel Ruiz Arreaza
Ciudad Real, Julio 2025

Resumen

«Que»

La Inteligencia Artificial está en ascenso, y muchas empresas están trabajando en la elaboración de una inmensa variedad de modelos que las ayuden tanto a ellas como a sus clientes a realizar una variedad de tareas igualmente amplia. Esto provoca que se realicen muchas entrenamientos de modelos al año y, en muchos casos, se creen o modifiquen muchos conjuntos de datos en el mismo intervalo de tiempo. Este es el caso de UBOTICA Tecnologías: una empresa de Space:AI que apuesta por la computación *on the edge* para ofrecer soluciones de Inteligencia Artificial integradas en sistemas empotrados en módulos espaciales, que tienen un espacio de almacenaje y poder computacional limitado. El desarrollo de estas soluciones requiere de varias iteraciones de entrenamiento y despliegue, lo que ha llevado a un desorden caótico de conjuntos de datos y modelos difícilmente identificables, lo que dificulta la búsqueda de un modelo o configuración de entrenamiento específica.

«Como»

MADTrack es un sistema de gestión de configuración distribuido que tiene el propósito de ponerle orden al caos configuracional previamente mencionado. El sistema almacenará, rastreará y gestionará todas las modificaciones de conjuntos de datos y configuraciones de modelos IA. Las principales restricciones que tiene el desarrollo del sistema son el espacio de almacenamiento limitado de la empresa para estos recursos (la evolución de conjuntos de datos y modelos debe ser registrada de manera eficiente), la naturaleza distribuida del entorno donde se almacenan y gestionan los elementos y la necesidad de la integración del sistema en un gran flujo de procesamiento. El desarrollo del sistema se dividirá en una serie de prototipos con un enfoque iterativo y incremental, siguiendo políticas de pruebas continuas.

«Conclusiones»

El sistema resultante consistirá en un sistema distribuido que sigue la arquitectura cliente-servidor, con un servidor local o remoto atendiendo las solicitudes de varios clientes enviando solicitudes a través de una biblioteca Software del tipo Wrapper que a su vez se interactúe con otras tecnologías de software abiertas.

Palabras clave: Escuela Superior de Informática, UCLM, TFG, L^AT_EX, T_EXLive, AI, Deep Learning, Distributed System, Dataset, configuration management .

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 deseas e incluir anécdotas y chascarrillos, pero recuerda que *no deberían ocupar más de una página*.

Miguel Ángel Ruiz Arreaza
Ciudad Real, 2025

Notación y acrónimos

NOTACION

(Texto aclaratorio -*suprime*-). Ejemplo de lista con notación (o nomenclatura) empleada en la memoria del TFG. Debes editarla según las necesidades de tu trabajo intenta que sea informativa y evita que incorpore información obvia.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)
 \bar{A} o A' : Operador NOT o negación

LISTA DE ACRÓNIMOS

(Texto aclaratorio -*suprime*-). Ejemplo de lista *ordenada alfabéticamente* con los acrónimos empleados en el texto. Se pueden omitir aquellos acrónimos que son reconocidos en el contexto académico (p. ej., PhD), aunque aquí se han incluido a efectos ilustrativos.

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

Índice general

Abstract	III
Resumen	IV
Agradecimientos	V
Notación y acrónimos	VI
Índice de figuras	IX
Índice de tablas	X
Listings	XI
List of Algorithms	XII
1. Introduction	1
1.1. Motivation	1
1.2. Problematic and Solution	2
1.3. Estructura del documento	3
2. Objetivo	4
2.1. El objetivo de un proyecto de ingeniería	4
3. Plan de gestión del trabajo	6
3.1. Guía rápida de las metodologías de desarrollo de software	6
3.2. Otros aspectos del plan de gestión	8
3.3. Tecnologías	8
4. Desarrollo y resultados	10
4.1. Requisitos y análisis del sistema	10
4.2. Diseño del sistema	10
4.3. Implementación del sistema	10
4.4. Pruebas del sistema	10
4.5. Despliegue	11
5. Conclusiones	12
5.1. Objetivos alcanzados	12
5.2. Justificación de competencias adquiridas	12

5.3. Trabajos derivados y futuros	12
5.4. Valoración personal	12
Bibliografía	13
A. Sobre la Bibliografía	15
B. Breve introducción a L^AT_EX	17
B.1. Listas	17
B.2. Tablas	18
B.3. Ecuaciones matemáticas	18
B.4. Figuras	19
B.5. Listados de código fuente	23
B.6. Menús, paths y teclas con el paquete menukeys	24

Índice de figuras

B.1. Ejemplo de figura	19
B.2. Ejemplo de subfiguras	20
B.3. Gráfico girado	21
B.4. Gráfico apaisado	22

Índice de tablas

3.1. Resumen de metodologías de desarrollo de software	7
B.1. Ejemplo de uso de la macro <code>cline</code>	18
B.2. Ejemplo de tabla con especificación de anchura de columna	18

Listings

B.1. Código fuente en Java	23
B.2. Ejemplo de código fuente en lenguaje C	23
B.3. Ejemplo de script en Matlab	23

List of Algorithms

B.1. Cómo escribir algoritmos	24
---	----

CAPÍTULO 1

Introduction

With the passage of time, the development of Artificial Intelligence (AI) has become of increasing interest due to the powerful tools it can provide to any organisation [3]. Since the development of *The Bombe*, the machine that was able to decode the *Enigma* machine in 1939, passing through a whole set of ups and downs and even a silent winter before its comeback in 2015 thanks to Deep Learning, Artificial Intelligence is being applied in a number of fields, sometimes even reaching the point of having the potential to threatening human safety and raising awareness of the need of regulations for its use.

The development of these models has been made possible thanks to the contribution of a number of organisations, such as Google, Microsoft and OpenAI, which invested an objectively significant amount of time and money (reaching a total investment of 24.0 billion dollars in 2018) to develop systems as broad as the GPT-4 model, which is able to generate text from both imaged and textual inputs [1], and is capable of helping professionals in solving doubts in a large number of fields.

With time, the development of these models became increasingly complex and difficult, and this would be reflected on the number of iterations required for a model to reach the desired level of accuracy in its predictions. Moreover, the application of these techniques on areas where little data was available made this development even harder. Some companies even started to employ its own resources for gathering new data to train their models, which led to a model having more iterations as the available data grew. Summarizing, a model (and even a dataset) can become complex structures that have many evolutionary stages within their lifecycle.

Many organisations, aware of the increasing complexity of the lifecycle of datasets and models, saw a market opportunity to develop systems that would manage these lifecycles in an organized and efficient way, producing the AI and dataset configuration management tools, such as MLFlow [13], DVC, which are open source and available to everyone, as well as premium tools such as Neptune.ai, which are paid Platform as a Service (PaaS) that provide even further monitoring capabilities.

The aim of this Final Degree Project is to tackle this issue in a specific particular case. In the coming subsections, the reasons that motivated the elaboration of this project, along with the challenges it faces and how will they be solved, will be thoroughly described. Finally, an overview of the structure of the document will be given, so as to give readers the necessary information to follow the development documentation of the project.

1.1. MOTIVATION

It is of common knowledge that Artificial Intelligence has gained significant importance in the 2020s. The development of tools such as the GPT models have headed a revolution in the way humans solve both simple and complex tasks. A revolution that was made possible with the intervention of

multiple companies and organisations, and a considerable amount of money and time invested in the development of these models [3].

UBOTICA Technologies is a pioneer company that develops **AI Computer Vision (CV)** solutions. This means that, as a company, they use Artificial Intelligence techniques to extract information from images. These solutions are then integrated in embedded systems with limited capabilities, and that are part of a bigger, more complex system. The domains where these solutions are used are mainly in the space industry with their new CogniSAT-6 project [8], and recently even in the culinary industry. The development of the various solutions of the company requires multiple iterations where the models are trained, validated and tested, either in existing datasets, or in new ones. Moreover, the CogniSAT-6 system has the capability for creating new datasets out of self-taken images, which may be periodically added to the existing datasets, or even used to replace the existing ones.

This continuous rise of the available models and datasets, paired with a lack of a real control protocol over the new and improved versions of an AI model or dataset, has led often to chaotic situations where an abnormal amount of time is taken on looking for the desired dataset, and accessing the necessary training configuration that produced an specific result on a model.

The aim of MADTrack is to develop a system that establishes a real configuration management basis for these datasets and AI models. A system that can integrate both new and existing datasets and models in a distributed, remote environment, and that will make the best use of the available resources the company dedicates to this management.

1.2. PROBLEMATIC AND SOLUTION

The main problematic to be tackled in this Final Degree Project is the lack of a real configuration management protocol over the produced datasets and models, the difficulty at not only determinating the correct configuration, but also bringing it to the environment where it is going to be used, and the need for establishing it from the methods used by the users so as to make the system familiar to them.

Furthermore, another problematic resides within the protocols and deployment requirements of the system, which may need special authentication protocols from the environment where the system is going to be deployed, as well as from the environment where the models and datasets are stored (which in turn may also differ from the one where the system is running in). All of this may result in the need to develop a distributed system, where many components interact with each other to satisfy a complex need. Moreover, the system must be easy to integrate in bigger workflows, so the study of the available pipelines the company uses and the possible integration points of the system within them may be taken as another problem.

Finally, the last problematic is the limitations of the company to dedicate resources for the storage of all the data produced by the model and dataset lifecycles. The developed system must, hence, make an efficient use of the resources available, so as to maximize the amount of data that could be stored and managed. Also, the system must be able to adapt to the needs of a growing company, where many requests may be taken concurrently.

For the resolution of all these three problems, the system will be formed by a server that will satisfy requests and perform the necessary storage and fetching operations to bring both the datasets and the models to the users, whilst the client will be a library that enables the integration of the necessary items into the configuration management, sending the evolutionary changes of a dataset or model to the server that stores them.

1.3. ESTRUCTURA DEL DOCUMENTO

Este capítulo suele finalizar con una sección en la que se indica la estructura (capítulos) del documento y el contenido de cada una de las partes en que se divide. Veamos a continuación cómo sería esta sección para este documento en concreto.

A lo largo de los capítulos que componen esta guía se muestran ejemplos de elementos de organización del texto en un documento preparado con \LaTeX . Los ejemplos mencionados, así como los recogidos en obras de referencia, se pueden emplear para adaptar este documento a las necesidades particulares [6, 12]. Entre las obras de consulta disponibles sobre \LaTeX se recomienda el uso de las obras gratuitas en español [7, 4] y las guías disponibles en la página web de [Overleaf](#) (en inglés).

En esta plantilla de TFG se ha optado por seguir la estructura orientativa que puede tener un TFG en la ESI-UCLM. Esta estructura consta de los capítulos siguientes:

1. **Introducción.** Donde se trata la motivación y la pertinencia del trabajo. Prosigue con el enunciado conciso del propósito del trabajo y la descripción de su contexto disciplinar y técnico.
2. **Objetivo.** En el que se detalla el alcance del objetivo general y los secundarios del trabajo.
3. **Plan de gestión del trabajo.** Describe la estrategia para abordar las distintas fases del trabajo.
4. **Desarrollo y resultados.** En este capítulo se explica cómo se han llevado a cabo las fases del trabajo cumpliendo el plan previsto y enumerando los resultados obtenidos.
5. **Conclusiones.** En el que se realiza una discusión sobre los resultados obtenidos y cómo estos satisfacen los objetivos planteados. Además, se justifica la aplicación al TFG de las competencias adquiridas durante los estudios de grado. También, puede incluir una explicación sobre los trabajos derivados y futuros si estos están planificados o iniciados, así como una breve valoración personal.
6. **Bibliografía.** Lista de las referencias bibliográficas que se hayan citado en el texto. Recuerda que no debes incluir fuentes de información relacionada que no hayas citado explícitamente en la memoria.
7. **Anexos.** Contenidos auxiliares que complementan del trabajo, como manuales de uso, diagramas, figuras, tablas, listados de código, etcétera.

CAPÍTULO 2

Objetivo

Este es un capítulo que explica en detalle el objetivo general del trabajo y los objetivos secundarios, si el principal admite una descomposición en módulos o componentes.

Es muy importante definir el objetivo de modo apropiado. Este 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 requisitos a considerar para la resolución del problema.

2.1. EL OBJETIVO DE UN PROYECTO DE INGENIERÍA

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:

- (A) 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*».
- (B) El *propósito académico* que la realización de un TFG tiene en la formación de un/a graduado/a. Por ejemplo, demostración de la adquisición de las competencias *específicas de la especialización* cursada y de aquellas *competencias transversales* ligadas a la realización del TFG.

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 se debe incluir en la memoria y en todo caso hacerse en la sección de conclusiones.*¹

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

- *Diseño y desarrollo de «artefactos».* Es un objetivo habitual en las ingenierías. Por la naturaleza de los programas informáticos (software), los trabajos que implican su diseño suelen contemplar también el desarrollo o implementación de prototipos. Esto es menos frecuente en otras áreas de ingeniería en las que claramente se separa la fase de diseño o realización de un proyecto, frente a la ejecución del mismo (p. ej., ingeniería civil, arquitectura, etc.).
- *Estudio* que ofrece información novedosa sobre un tema (usual en las ramas de ciencias y humanidades).

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

- *Validación de una hipótesis* de partida. Este tipo de objetivo es propio de los trabajos científicos, pero 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 cualquier caso, cuando el objetivo sea lo suficientemente amplio es conveniente su descomposición en elementos más simples hablando de *objetivos secundarios* o *subobjetivos*. Por ejemplo, un programa informático se podría descomponer 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 debe 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 las *restricciones* que acotan su alcance. En el caso de un trabajo cuyo objetivo es el desarrollo de un «artefacto» o sistema, los requisitos pueden ser *funcionales* (qué debe hacer el sistema) y *no funcionales* (cómo debe ser el sistema).

Al redactar el objetivo de un TFG se puede confundir los medios con el fin. De este modo a veces se define un objetivo en términos de las *acciones* (verbos) o *tareas* necesarias para alcanzarlo. Sin embargo, aunque no definen el objetivo, es apropiado descomponer este en *hitos* y *tareas* para facilitar la *planificación* del trabajo.

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

CAPÍTULO 3

Plan de gestión del trabajo

En este capítulo se debe detallar todos los aspectos relacionados con el plan de gestión del trabajo que incluyen:

- la metodología de desarrollo,
- las tecnologías y recursos necesarios,
- la gestión de la configuración y el aseguramiento de la calidad,
- la planificación del trabajo, y
- la estimación de costes y el análisis de riesgos.

3.1. GUÍA RÁPIDA DE LAS METODOLOGÍAS 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 ingenieros 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 sea 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.

3.1.1. Metodologías de desarrollo software

Las metodologías son el modo en que las fases del proceso de desarrollo de 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:

- A. **Procedimientos** (indican cómo hacer cada tarea y en qué momento),
- B. **Herramientas** (ayudas para la realización de cada tarea), y
- C. **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 [11, 2].

Mientras que unas metodologías 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 resume a continuación en la tabla 3.1.

Tabla 3.1: Resumen de metodologías de desarrollo de software

Metodología	Características	Herramientas/Enfoques asociados
SSADM <i>(Structured Systems Analysis and Design Methodology)</i>	Metodología estructurada, secuencial (modelo en cascada), centrada en análisis exhaustivo de requisitos.	Modelado lógico de datos, flujo de datos, entidades y eventos.
OOD <i>(Object-Oriented Design)</i>	Centrado en la modularidad y reutilización del código a través de objetos y clases.	UML (Diagramas de clase, casos de uso, secuencia, modelo de datos).
RAD (<i>Rapid Application Development</i>)	Desarrollo iterativo y rápido basado en prototipos, minimiza documentación, enfatiza entrega rápida.	Interfaces gráficas, prototipado, ciclo en espiral (Boehm), evaluación continua por parte del usuario.
Metodologías Ágiles	Desarrollo incremental e iterativo, énfasis en la colaboración y en software funcional, poca documentación.	<i>Scrum, Kanban</i> , reuniones diarias, sprints, enfoque en el concepto de “hecho (done)”.

3.1.2. Proceso de testing

El testeo del software puede consistir en varios tipos de procesos:

1. *Pruebas modulares* (pruebas unitarias). Su propósito es hacer pruebas sobre un módulo tan pronto como sea posible. Las *pruebas unitarias* comprueban el correcto funcionamiento de una unidad de código. En la programación estructurada, dicha unidad elemental de código consistiría en cada función o procedimiento. Para la programación orientada a objetos se

trataría de cada clase. 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.*

3.2. OTROS ASPECTOS DEL PLAN DE GESTIÓN

Además de la metodología se deben abordar los aspectos siguientes relacionados con el plan de gestión del proyecto:

- **Recursos.** En este apartado se describirán los recursos hardware y software empleados. También debería quedar aclarado el número y papel de los integrantes del equipo de proyecto.
- **Gestión de la configuración y aseguramiento de la calidad.** Durante el desarrollo de proyectos de software es esencial definir la estrategia de control de versiones y las diferentes *releases*. Además, en esta sección se describirán las actividades y tareas que garantizan la calidad del proceso de desarrollo del software, incorporando los estándares, prácticas y normas de aplicación. También se deben documentar los distintos tipos de revisiones, verificaciones y validaciones que se realizarán, criterios de aprobación o rechazo de cada, y los procedimientos para llevar a cabo acciones correctivas o preventivas.
- **Planificación.** Detallará la estimación de la evolución temporal del proyecto, marcando sus iteraciones e hitos básicos. Para ello, se emplearán diagramas Gantt y debería incluir una comparación cuantitativa del tiempo y el esfuerzo realmente invertido frente al estimado.
- **Costes.** Análisis y presupuesto del coste de los recursos (humanos y materiales) necesarios para el proyecto. El cálculo de costes de personal debe tener en cuenta la realidad del mercado laboral en España. Dicho cálculo se puede hacer en persona/mes, y luego hacer la correspondencia al coste monetario.
- **Análisis de riesgos.** En esta sección se debería incluir una enumeración de los riesgos del proyecto, indicando su posible impacto (efecto que la ocurrencia del citado riesgo tendría en el desarrollo del proyecto) y la probabilidad de ocurrencia. Una vez se identifican los riesgos, se deben priorizar para definir los planes necesarios que reduzcan su impacto o incluso su probabilidad de ocurrencia.

3.3. REVISIÓN DE TECNOLOGÍAS Y HERRAMIENTAS CASE (COMPUTER AIDED SOFTWARE ENGINEERING)

Además de los recursos humanos y de hardware necesarios en el trabajo, en la sección de recursos software se deberían enumerar las herramientas software previstas. A continuación se realiza una revisión rápida no exhaustiva de las herramientas más populares.¹

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 las siguientes categorías:

1. Modelado y análisis de negocio.
2. Desarrollo.

¹Este listado, no exhaustivo, es meramente informativo, ya que en la memoria de un TFG solo se deben incluir aquellas que se hayan evaluado y empleado finalmente.

3. Verificación y validación.
4. Gestión de configuraciones.
5. Métricas y medidas.
6. Gestión de proyecto (gestión de planes, asignación de tareas, planificación, etc.).

3.3.1. IDE (Integrated Development Environment)

- [Visual Studio Code](#)
- [Atom](#)
- [GNU Emacs](#)
- [NetBeans](#)
- [Eclipse](#)
- [Qt Creator](#)
- [jEdit](#)
- [IntelliJ IDEA](#)

3.3.2. Depuración

- [GNU Debugger](#)

3.3.3. Testing

- [JUnit](#). Entorno de pruebas para Java.
- [CUnit](#). Entorno de pruebas para C.
- [PyUnit](#). Entorno de pruebas para Python.
- [NUnit](#). Entorno de pruebas para .Net.

3.3.4. Repositorios y control de versiones

- [Git](#)
- [Github](#)
- [Mercurial](#)
- [Bitbucket](#)
- [SourceTree](#)

3.3.5. Documentación

- [L^AT_EX](#)
- [Overleaf](#)
- [Markdown](#)
- [Doxygen](#)
- [DocGen](#)
- [Pandoc](#)

3.3.6. Gestión y planificación de proyectos

- [Trello](#)
- [Jira](#)
- [Asana](#)
- [Slack](#)
- [Basecamp](#)
- [Teamwork Projects](#)
- [Zoho Projects](#)

CAPÍTULO 4

Desarrollo y resultados

En este capítulo se describirá las diferentes fases del ciclo de desarrollo de software de acuerdo al plan de gestión expuesto en el capítulo 3.

4.1. REQUISITOS Y ANÁLISIS DEL SISTEMA

En esta sección se presentarán los objetivos y el catálogo de requisitos del proyecto: funcionales, no funcionales, de información, reglas de negocio, etc. Una vez catalogados los requisitos del sistema se procederá con su análisis empleando el lenguaje de modelado UML. El análisis mencionado incluirá:

- **Modelo conceptual.** En él se identifican clases, atributos, relaciones, etc.
- **Modelo de casos de uso.** Representan las interacciones entre los actores y el sistema bajo estudio.
- **Modelo de interfaz de usuario.** Puede consistir en un prototipo de baja fidelidad de la interfaz de usuario.

4.2. DISEÑO DEL SISTEMA

En esta sección se define la arquitectura lógica general del sistema. Para describirla se puede emplear un modelo como C4 [5], el cual permite representar la arquitectura de un sistema software mediante varios diagramas a distintos niveles de abstracción.

4.3. IMPLEMENTACIÓN DEL SISTEMA

En este apartado se describirá la organización del código fuente y *scripts*, describiendo el propósito de los distintos ficheros y su distribución en paquetes y directorios. Puede ser conveniente incluir alguna porción significativa de código fuente que sea de especial relevancia por su funcionalidad.

En el desarrollo de proyectos software cobra especial importancia el empleo de sistemas de control de versiones junto a repositorios en línea. Estas herramientas se convierten en esenciales para disponer de un registro histórico del desarrollo que también puede ayudar a evaluar el trabajo realizado. Por este motivo, en la memoria del proyecto se debe indicar la dirección URL de los repositorios empleados (p. ej., Github).

4.4. PRUEBAS DEL SISTEMA

En esta sección se describirá el plan de pruebas del sistema incluyendo todos los tipos llevados a cabo. El desarrollo de la sección debería tratar los aspectos siguientes:

- **Estrategia.** Donde se indica el alcance de las pruebas y los procedimientos.

- **Pruebas unitarias.** Destinadas a localizar errores en cada nuevo módulo software desarrollado antes de su integración con el resto del sistema.
- **Pruebas de integración.** Su objetivo es localizar errores en subsistemas completos analizando la interacción entre varios módulos de software.
- **Pruebas de sistema.** Contempla las pruebas funcionales con las que se realiza el análisis del buen funcionamiento de la implementación de los casos de uso del sistema. Además, en estas pruebas se comprueba el funcionamiento respecto a los requisitos no funcionales: eficiencia, seguridad, etcétera.
- **Pruebas de aceptación.** Su intención es demostrar, con la participación del cliente, que el producto está listo para su puesta en funcionamiento en el entorno producción.

4.5. DESPLIEGUE

Esta sección recoge la arquitectura física propuesta del sistema, las instrucciones para su despliegue, operación y mantenimiento del nivel de servicio. 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, etcétera.

CAPÍTULO 5

Conclusiones

5.1. OBJETIVOS ALCANZADOS

En este capítulo se realizará un juicio crítico y discusión sobre el objetivo general y objetivos secundarios alcanzados durante el desarrollo del trabajo.

5.2. JUSTIFICACIÓN DE COMPETENCIAS ADQUIRIDAS

Es muy importante recordar que según la normativa vigente en la ESI-UCLM, el capítulo de conclusiones debe incluir *obligatoriamente* un apartado destinado a justificar la aplicación en el TFG de las competencias específicas (una o más) adquiridas en la tecnología específica cursada, como se indica a continuación:

En el TFG se han aplicado 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 se ha aplicado en el TFG.

...(otras más si las hubiera).

5.3. TRABAJOS DERIVADOS Y FUTUROS

Si es pertinente se puede 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*).

Se recomienda reflexionar sobre la conveniencia de inclusión de una lista de posibles mejoras, ya que puede transmitir la impresión de que el trabajo se encuentra en un estado incompleto o inacabado.

5.4. VALORACIÓN PERSONAL

En esta sección final se realizará un rápido análisis de las lecciones aprendidas en las que se pueden incluir tanto buenas prácticas adquiridas (tecnológicas y procedimentales) como cualquier otro aspecto de interés. También se puede resumir cuantitativamente el tiempo y esfuerzo dedicados al proyecto a lo largo de su desarrollo.

Bibliografía

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Brook Appelbaum. Top 6 software development methodologies. URL: <https://blog.planview.com/top-6-software-development-methodologies>, September 2022. Último acceso 26 feb. 2024.
- [3] Jasmin Praful Bharadiya, Reji Kurien Thomas, and Farhan Ahmed. Rise of artificial intelligence in business and industry. *Journal of Engineering Research and Reports*, 25(3):85–103, 2023.
- [4] Alexánder Borbón and Walter Mora. *Edición de textos científicos con LATEX. Composición, diseño editorial, gráficos, Inkscape, Tikz y presentaciones Beamer*. Instituto Tecnológico de Costa Rica, 2 edition, 2021.
- [5] Simon Brown. The C4 model for visualising software architecture. URL: <https://c4model.com/>, 2022. Último acceso: 19 feb. 2024.
- [6] Leslie Lamport. *LATEX: A document preparation system*. Addison-Wesley, second edition, June 1994. ISBN: 978-0201529838.
- [7] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *La introducción no-tan-corta a LATEX2e*, 2014. URL: <http://www.ctan.org/tex-archive/info/lshort/spanish/>.
- [8] David Rijlaarsdam, Tom Hendrix, Pablo T Toledo González, Alberto Velasco-Mata, Léonie Buckley, Juan Puig Miquel, Oriol Aragon Casaled, and Aubrey Dunne. The next era for earth observation spacecraft: An overview of cognisat-6. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.
- [9] Servicio de publicaciones UCLM. Propiedad intelectual. Documentos elaborados por el Grupo de Gestión del Conocimiento y Propiedad Intelectual de la Universidad de Castilla-La Mancha. URL: <https://e.uclm.es/servicios/doc/?id=UCLMDOCID-12-739>. Último acceso: feb. 2024.
- [10] Universidad de Cantabria. Cómo usar imágenes en trabajos. Artículo técnico disponible en URL: https://web.unican.es/buc/Documents/Formacion/guia_imagenes.pdf, 2018. Último acceso: sep. 2021.
- [11] Leo R. Vijayasarathy and Charles W. Butler. Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter? *IEEE Software*, 33(5):86–94, September 2016. DOI: <https://doi.org/10.1109/ms.2015.26>.
- [12] WikiMedia. LATEX Wikibook. URL: <http://en.wikibooks.org/wiki/LaTeX>, 2010. Último acceso: sep. 2021.
- [13] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

APPENDICES

APÉNDICE A

Sobre la Bibliografía

En los anexos se incluirá, de modo opcional, material suplementario que podrá consistir en manuales de usuario, listados seleccionados de código fuente, esquemas, planos y en general aquel contenido que complementa a la memoria. Se recomienda que no sean excesivamente voluminosos, aunque su extensión no está sometida a la regulación por normativa, ya que esta afecta únicamente al texto principal de la memoria.

En esta plantilla hemos decidido incluir dos anexos. En el primero de ellos se hacen algunos comentarios adicionales sobre la bibliografía. En el segundo se aporta una breve introducción a L^AT_EX cuya información puede servir de ejemplo de inclusión de ciertos elementos en la preparación de la memoria del TFG.

Todo el material de terceros se debe citar convenientemente sin contravenir los términos de las licencias de uso y distribución de dicho material. Esto se extiende al uso de diagramas y cualquier elemento gráfico. 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.¹

La sección de *Bibliografía*, que si se prefiere se puede titular *Referencias*, incluirá un listado ordenado preferentemente por orden alfabético (primer apellido del autor principal), con todas las obras citadas en el texto. En la lista de referencias se especificará para cada obra:

- autores,
- título,
- editorial, y
- año de publicación.

Este formato se conseguirá en L^AT_EX mediante el uso del estilo estándar plain o cualquier otro derivado con estilo de citación numérica. En algunas titulaciones se obliga a una ordenación por orden de cita en el texto. Este resultado se obtiene con BibT_EX mediante los estilos estándar ieeetr (estilo para los IEEE *transactions*) y unsrt (estilo *unsorted*).²

Es muy importante tener presente que en esta sección solo se debe incluir las referencias bibliográficas citadas expresamente en el documento. Si se desea incluir fuentes consultadas, pero no citadas, se puede confeccionar con ellas una sección denominada *Material de consulta*, aunque estas referencias se pueden incluir opcionalmente a lo largo del documento como notas a pie de página.

En las titulaciones técnicas se empleará estilo de citación numérico con el número de la referencia entre corchetes. La cita podrá incluir el número de página concreto de la referencia que se desea citar.

¹<https://www.uclm.es/areas/biblioteca/encuentra-informacion/perfiles/alumno/antiplagio>

²<https://www.wikiwand.com/es/articles/BibTeX>

El uso correcto de la citación implica dejar claro al lector cuál es el texto, material o idea citado. .

Cuando se desee incluir referencias a páginas genéricas de Internet sin mención expresa a un artículo con título y autor definido, dichas referencias se pueden incluir como notas al pie de página o como un apartado de fuentes de consulta dedicado a *Direcciones de Internet*. Por el contrario, los documentos electrónicos publicados en Internet se pueden incluir en la sección de Bibliografía empleando el tipo de entrada `misc` en el fichero `.bib` con el comando `url` (como se muestra en la bibliografía de ejemplo distribuida con este documento). Observarás que el campo `note` se emplea para añadir información adicional como la fecha de la última consulta de fuentes publicadas en Internet, y para la inclusión del DOI de algunas obras para su rápida recuperación.³

³Esta estrategia necesita adaptación cuando se emplea un estilo de citación autor-año (p.ej., `natbib`).

APÉNDICE B

Breve introducción a L^AT_EX

El contenido del trabajo final de estudios se organiza en capítulos que se subdividen en secciones. Con L^AT_EX este tipo de organización se realiza de modo inmediato mediante la generación automática de los estilos correspondientes a los títulos de cada sección y su inclusión en la tabla de contenidos. Los ajustes relativos a la generación del formato y estilos asociados a secciones del documento se realizan con el paquete `titlesec` empleado en esta plantilla.

En las secciones siguientes se comenta la inclusión con L^AT_EX de distintos elementos de organización de información junto a ejemplos que facilitan su utilización en la memoria del trabajo.¹

B.1. LISTAS

Existen dos tipos de listas: enumeraciones y listas con viñetas. En el primer tipo, los elementos de la lista se preceden de una clave numérica o alfabética, mientras que en el segundo tipo se emplea una viñeta. En ambos casos los elementos se pueden anidar para crear una jerarquía entre ellos. En L^AT_EX se recomienda la inclusión del paquete `enumitem` (incluido en esta plantilla) que permite personalizar fácilmente las listas de un documento. A continuación se muestran algunos ejemplos de las posibilidades.

B.1.1. Ejemplo de lista con viñetas personalizadas

- pera
- ☛ manzana
- ❶ naranja

B.1.2. Ejemplo de lista condensada

Este tipo de lista presenta una separación mínima entre ítems, en varias columnas y configuración de la etiqueta.

- | | |
|-------------|--------------|
| (1) pera | (4) patata |
| (2) manzana | (5) calabaza |
| (3) naranja | (6) fresa |

Además del texto, los documentos pueden incluir elementos que enriquecen su contenido facilitando su exposición y comprensión. En las secciones siguientes tratamos brevemente dichos elementos.

¹Las explicaciones de este anexo forman parte del contenido del curso «L^AT_EX esencial para preparación de TFG y otros documentos académicos» de la ESI-UCLM.

B.2. TABLAS

A continuación se incluyen algunos ejemplos de tablas elaboradas con L^AT_EX mediante el empleo de paquetes dedicados. Para la realización de tablas más complejas se recomienda la consulta de manuales [4] y el empleo de asistentes o herramientas en línea.²

Observa que el título de las tablas se ubica en la parte superior de la tabla. Puesto que el contenido de la tabla es texto, tiene sentido leer primero el título para contextualizar el contenido de la tabla antes de su lectura.

Tabla B.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 B.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

B.3. ECUACIONES MATEMÁTICAS

La composición de ecuaciones requiere el uso de comandos especializados. Por tanto, para facilitar dicha tarea se aconseja el empleo de programas especializados como MathType o asistentes como el incluido en editores como T_EXstudio³ o herramientas en línea.⁴ Es muy simple incluir fórmulas matemáticas sencillas en el mismo texto en el que se escribe. Por ejemplo, $h^2 = a^2 + b^2$ que podría ser la ecuación representativa del teorema de Pitágoras (ver también ec. B.1).

²<https://www.tablesgenerator.com/>

³<https://www.texstudio.org/>

⁴<https://latex.codecogs.com/>, <http://www.sciweavers.org/free-online-latex-equation-editor>

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

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

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

$$h^2 = b^2 + c^2 \tag{B.1}$$

B.4. FIGURAS

A diferencia de lo que sucede en las tablas, el título de las figuras aparece en la parte inferior de estas. Para la inclusión de las figuras se debe tener en cuenta que su contenido se encuentra en un fichero individual con el formato y resolución apropiados para garantizar la calidad del resultado final.

En esta sección se añaden ejemplos de muestra para la inclusión de figuras simples y otras compuestas de subfiguras mediante el empleo del paquete `subcaption`.



Figura B.1: Fotografía a color (Fuente: J. Salido, CC BY-NC-ND)

Ejemplo de figura compuesta por dos subfiguras incluidas mediante paquete `subcaption`. A través del uso de etiquetas (`\label`) es posible incluir referencias cruzadas a subfiguras como la fotografía en blanco y negro de la Fig. B.2b.



(a) Fotografía a color



(b) Fotografía en blanco y negro

Figura B.2: Ejemplo de inclusión de subfiguras en un mismo entorno (Fuente: J. Salido, CC BY SA)

En los trabajos académicos la inclusión de imágenes y figuras que no son propiedad del autor suscitan bastante controversia, ya que con frecuencia se incumple inadvertidamente la ley vigente de propiedad intelectual. Respecto a este hecho se recomienda, tanto al alumnado como al profesorado encargado de la tutorización, consultar documentación informativa sobre el uso correcto de figuras en documentos académicos [9, 10]. Entre las «incorrekiones» más habituales en los documentos académicos, se observa:

- *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).
- *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. B.1 y B.2 mencionando al autor y la licencia Creative-Commons⁵ 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 usando título opcional.
- *Supresión de los detalles de la 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 autoría.

La inclusión de material de *dominio público*, sin restricciones de uso o con permiso, hace innecesaria la atribución al autor, pero se recomienda incluir una nota de agradecimiento.⁶

Cuando se presenta la necesidad de incluir un gráfico demasiado grande para el tamaño de la página, una opción muy apropiada es la impresión del gráfico en modo girado en una página aparte. Este efecto se consigue con el entorno `sidewaysfigure` proporcionado por el paquete `rotating`. La Fig. B.3 muestra un ejemplo del entorno citado con un gráfico PDF.

⁵<https://creativecommons.org>

⁶Incluyendo un texto como: «Por cortesía de ...»

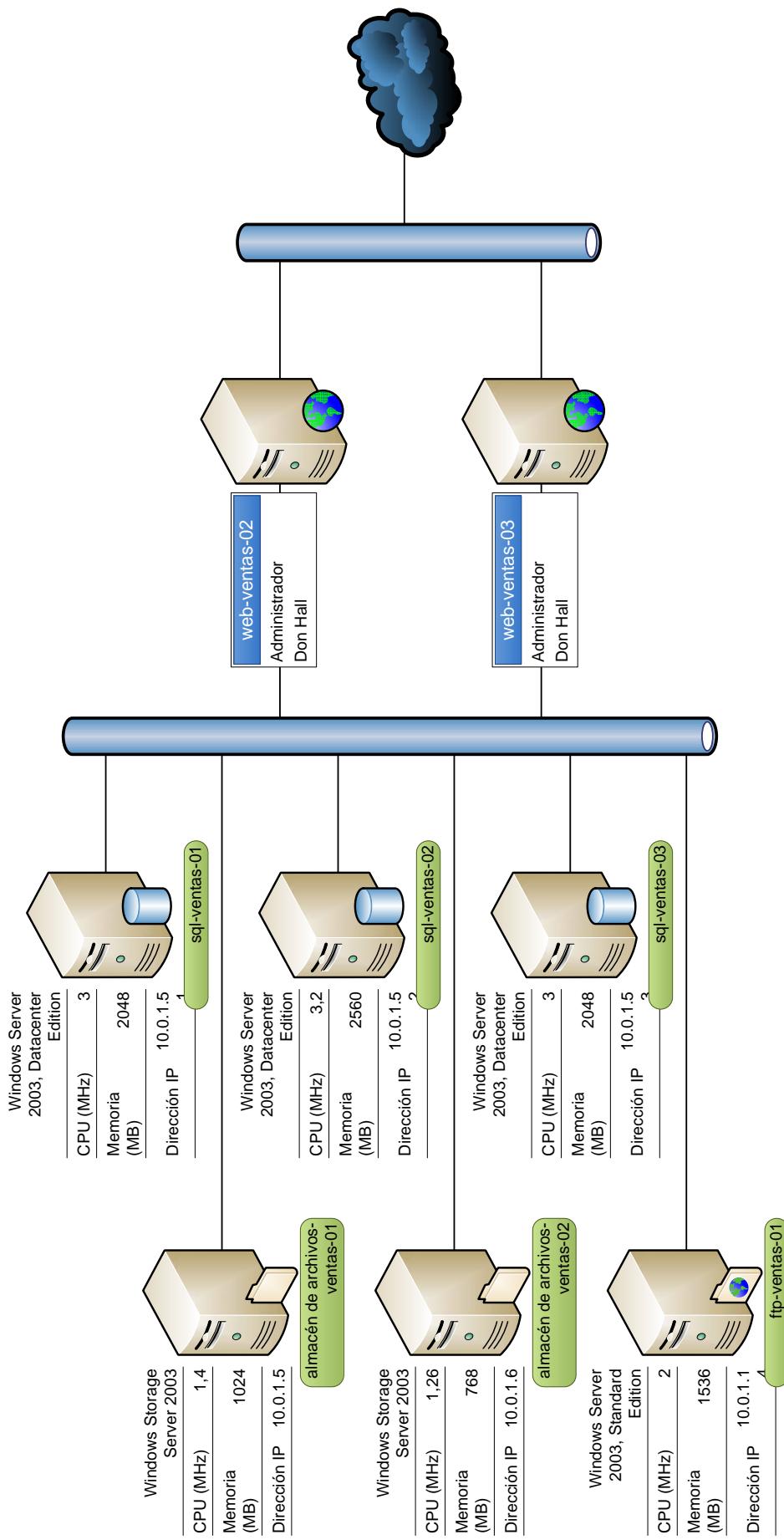


Figura B.3: Figura vectorial con impresión girada

También es posible imprimir una página en formato apaisado cuando contiene una figura muy ancha. Este efecto se consigue con el paquete `pdfescape` y el entorno `landscape` proporcionado. Además, en este caso se han suprimido tanto la cabecera como el pie de página. La figura B.4 se muestra apaisada a modo de ejemplo.

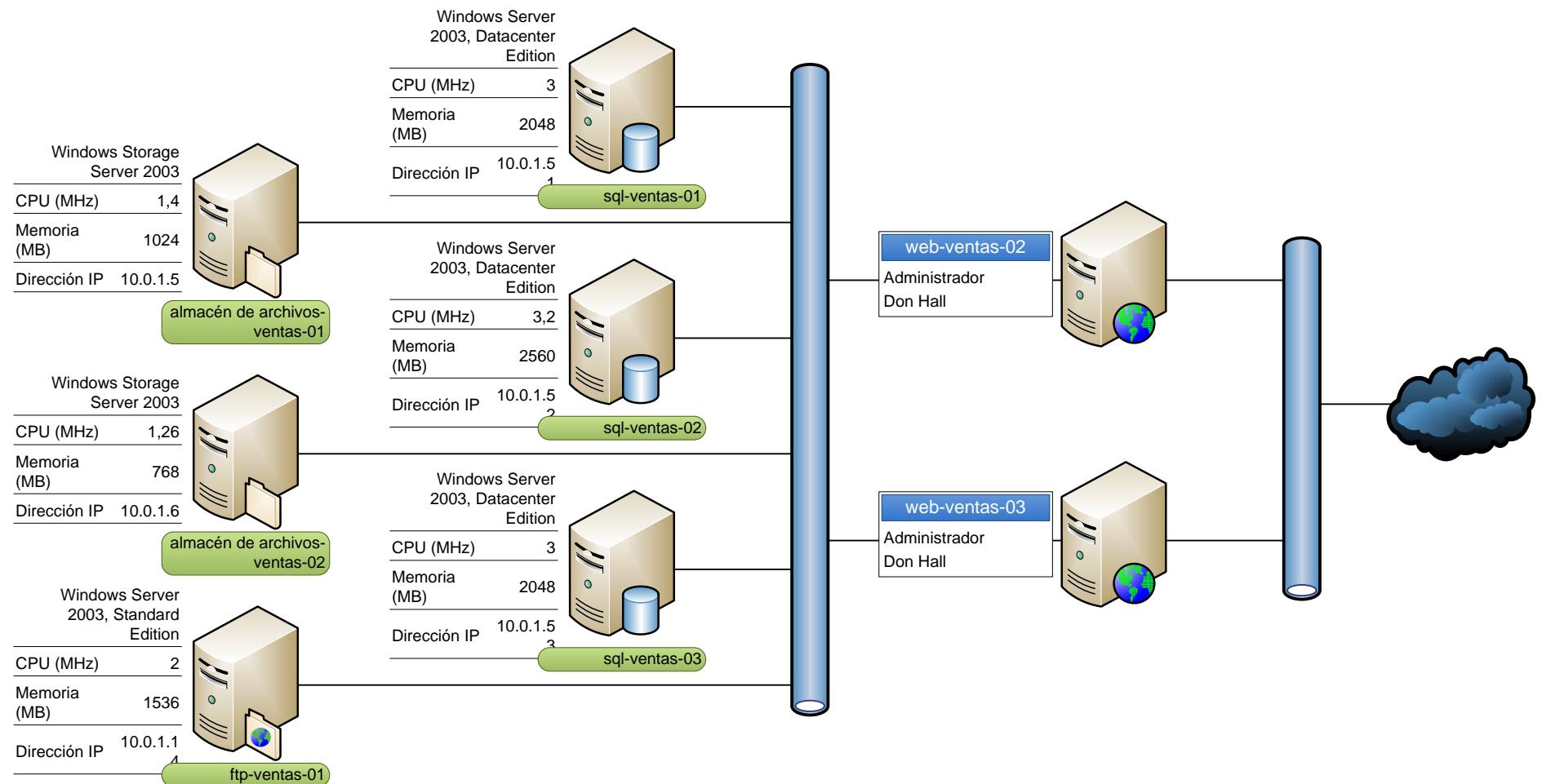


Figura B.4: Figura vectorial con vista en página apaisada

B.5. LISTADOS DE CÓDIGO FUENTE

En los textos científicos relacionados con las TIC⁷ (Tecnologías de la Información y Comunicaciones) suelen aparecer porciones de código en las que se explica alguna función o característica relevante del trabajo que se expone.

La inclusión de porciones de código fuente se puede formatear de modo sencillo en LATEX mediante el uso del paquete `listings`. A continuación, se muestran varios ejemplos de porciones de código correspondientes a distintos lenguajes de programación.

Listing B.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\u00a0String\u00a0is\u00a015";
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 }
```

Listing B.2: Ejemplo de código fuente en lenguaje 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\u00a0mundo!");
6 }
```

Listing B.3: Ejemplo de script en Matlab

```

1 function f = fibonacci(n)
2 % FIBONACCI Fibonacci sequence
3 % f = FIBONACCI(n) generates the first n Fibonacci numbers.
4 % Copyright 2014 Cleve Moler
5 % Copyright 2014 The MathWorks, Inc.
6
7 f = zeros(n,1);
8 f(1) = 1;
9 f(2) = 2;
10 for k = 3:n
11     f(k) = f(k-1) + f(k-2);
12 end
```

Algunas veces lo que se quiere ilustrar es un algoritmo o método con el que se resuelve un problema abstrandose del lenguaje de implementación. El paquete `algorithm2e` proporciona un entorno `algorithm` para la impresión apropiada de algoritmos, tratándolos como objetos flotantes y con mucha flexibilidad de personalización, como se observa en el algoritmo B.1 del ejemplo.

⁷Por supuesto, en un TFG (Trabajo Fin de Grado) o tesis de un centro superior de Informática.

Algorithm B.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

```

B.6. 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 de menús, pulsaciones de teclas y directorios, se recomienda el uso del paquete menukeys.⁸ 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, esto no es necesario para obtener unos resultados muy elegantes.

⁸<https://osl.ugr.es/CTAN/macros/latex/contrib/menukeys/menukeys.pdf>