Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño de una aplicación para la gestión de documentos

Proyecto Final de Carrera

Ingeniería técnica en informática de gestión

Autor: Alberto Montero Mañez

Director: Oscar Sapena Vercher

01/09/2013

Índice general

Índice de figuras

1 Introducción

En este capítulo se expone el trabajo que se va a realizar en este proyecto empezando por un análisis, una motivación y descripción del problema que se va a abordar. Una vez presentado el problema veremos cuál es la situación actual y las diferentes alternativas a ese problema. Adicionalmente, se presenta el objetivo del proyecto y los beneficios que se obtendrán.

1.1 Introducción y motivación.

La información que manejan las empresas aumenta diariamente (pedidos, albaranes, facturas, expedientes, proyectos, catálogos, normas y procedimientos, etc.). Es común a todas las empresas, sin importar su tamaño, la problemática de gestionar su documentación.

La gestión documental es la manipulación eficiente de todo ese volumen de contenidos en formato electrónico, con el objetivo de reducir costes, espacio y obtener acceso inmediato a toda la información.

"La oficina sin papel" fue la idea que impulsó la implantación de los primeros sistemas de gestión documental en las empresas. El aumento de los metros cuadrados dedicados al almacenamiento de papel y del tiempo invertido en la búsqueda y consulta de documentos, hicieron que muchas organizaciones se decidieran a invertir en plataformas electrónicas.

Desde ese momento, la gestión documental ha experimentado una continua evolución en su afán de adaptarse a las necesidades cambiantes de las empresas y, a día de hoy, se centra en la gestión de documentos que ya nacieron en formato electrónico, en la

extracción automática de los datos que contienen y en el traspaso de estos datos al sistema ERP (Enterprise Resource Planning, sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o servicios).

1.2 Descripción del problema.

La idea surge de pensar en el volumen de información que almacena una empresa y como poder agilizar la clasificación de dicha información.
Si conseguimos pasar toda esta información a un sistema informático podemos automatizar el proceso y reducir el tiempo y los recursos para almacenar los documentos que deseemos.

Imagen 1. Cantidad de documentos

1.3 Objetivos del proyecto.

El objetivo principal del proyecto es desarrollar una aplicación, que además de proporcionar una interfaz visual rápida y de sencillo manejo, haga más eficiente todo lo relacionado con las tareas de almacenamiento de documentos. Los principales elementos sobre los que se trabaja y, por tanto, sobre los que se desea guardar información son los siguientes:

- Carpetas de entrada. Estas serán las carpetas donde se depositarán los documentos y se almacenaran en el valor correspondiente.

- Categorías. Las categorías serán el tipo de documento a almacenar.
- Valores. En estos se almacenan los documentos según a la empresa que representan.
- Trabajos. Estos se encargan de estar revisando continuamente las carpetas de entrada para ver si tienen nuevos archivos y los revisan con las categorías que tengan asignadas para almacenarlos correctamente.

La creación de este sistema informático proporcionará unos beneficios obvios. Es evidente que la utilización de una aplicación de gestión proporcionará una mejora en cuanto a rendimiento, velocidad y facilidad de acceso que se puede obtener.

1.4 Beneficios.

El desarrollo de esta aplicación y su posterior uso proporcionarán al taller los beneficios que se exponen a continuación:

- Ahorrar tiempo. El tiempo de gestión y consulta se reduce drásticamente. Los documentos electrónicos son notablemente más fáciles de editar, reutilizar y/o enviar. Y, lo que es aún más importante, se minimiza la posibilidad de extravío o pérdida de documentos.

- Ahorrar dinero. Está comprobado que de gestionar en papel a gestionar en electrónico se obtiene un ahorro de entre un 50 y un 70% en:
  - Papel, toners, fotocopias, faxes, etc.
  - Sellos, sobres, etc.
  - Etiquetas, carpetas, armarios archivadores, m2.
  - Horas de personal.

- Mejorar la seguridad. Se controla el acceso a la información facilitando el cumplimiento de la LOPD (Agencia Española de Protección de Datos).

Estas tareas harán que las tareas administrativas sean más rápidas y eficientes y mejorarán la productividad notablemente.

2 Marco teórico.

En este capítulo se exponen los conocimientos teóricos que se han utilizado para la realización de este proyecto y se plantea la forma de abordar el problema desde un paradigma orientado a objetos. Los conocimientos teóricos son: programación orientada a objetos, modelado conceptual, diseño conceptual, diseño de interfaces gráficas de usuario, diseño de bases de datos y desarrollo de componentes.

2.1 Programación orientada a objetos.

La programación orientada a objetos [2, 12] también conocida por las siglas POO surge como un nuevo paradigma de programación a mediados de los 80. En la actualidad se ha vuelto muy popular porque la forma de percibir y pensar sobre los problemas se acerca a los mecanismos cognitivos humanos y es muy parecida a la forma en que se abordan los problemas en otros dominios. Existe la esperanza de que esta técnica sea la clave para incrementar la productividad y mejorar la fiabilidad de los sistemas software [9].

En la programación estructurada el programador piensa en términos de procedimientos y en los datos que manejan esos procedimientos. La programación orientada a objetos, en cambio, define los programas en términos de "clases de objetos". Los objetos son instancias (entidades) de una clase que tienen identidad propia (se distinguen unos de otros), comportamiento (pueden realizar ciertas tareas) y estado (almacenan información).

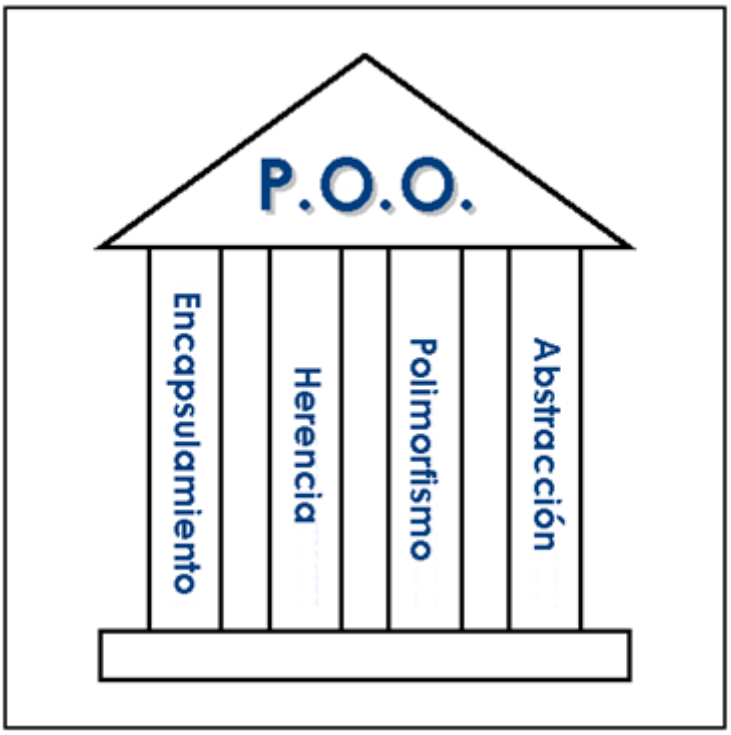

Imagen 2. Programación Orientada a Objetos

Desde el punto de vista del analista, una clase es un elemento/ente/artefacto de clasificación humana basada en un comportamiento y/o propiedades comunes; desde el punto de vista del programador se trata de una construcción sintáctica que describe comportamiento y atributos comunes. La programación orientada a objetos se expresa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esto es diferente de los lenguajes de programación estructurada tradicionales, en los que los procedimientos y los datos están separados y sin ninguna relación. La metodología "orientada a objetos" está pensada para hacer los programas y módulos más fáciles de entender, reutilizar y mantener. Dicho de otra manera, la programación orientada a objetos permite al programador pensar en los programas principalmente en términos de tipos de datos, y en las operaciones ("métodos") específicas a esos tipos de datos. Por tanto las características más importantes de la programación orientada a objetos son las siguientes:

- **Encapsulación**: (también conocida como "ocultación de la información"). Cada objeto expone una interfaz a otros

objetos, que especifica cómo los otros objetos pueden interactuar con él, a través de sus métodos públicos. Por otra parte, el objeto dispone de una sección privada en la que se encuentran, entre otras, los datos que maneja internamente, para llevar a cabo su tarea. Estos datos no son accesibles por los otros objetos. Esta característica de la POO nos asegura que los objetos no pueden modificar el estado interno de otros objetos de forma inesperada, y si se producen cambios en el tipo de datos de la parte privada, la utilización del objeto no se ve afectada.

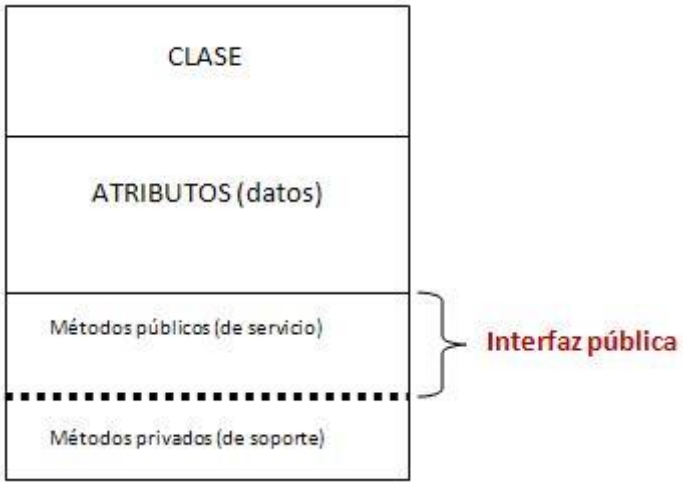

Imagen 3. Encapsulación

- **Herencia:** es el principio por el cual se dice que el conocimiento de una clase más general es aplicable también a la clase más específica. La herencia permite que diferentes tipos de datos compartan el mismo código, lo que conduce a una reducción (por reutilización) del tamaño del código y a un incremento en su funcionalidad. Las clases se pueden organizar en una estructura de árbol, llamada jerarquía de herencia. La información (datos y comportamiento) asociados con un nivel de una jerarquía de clases es aplicable automáticamente a los niveles inferiores de la jerarquía.

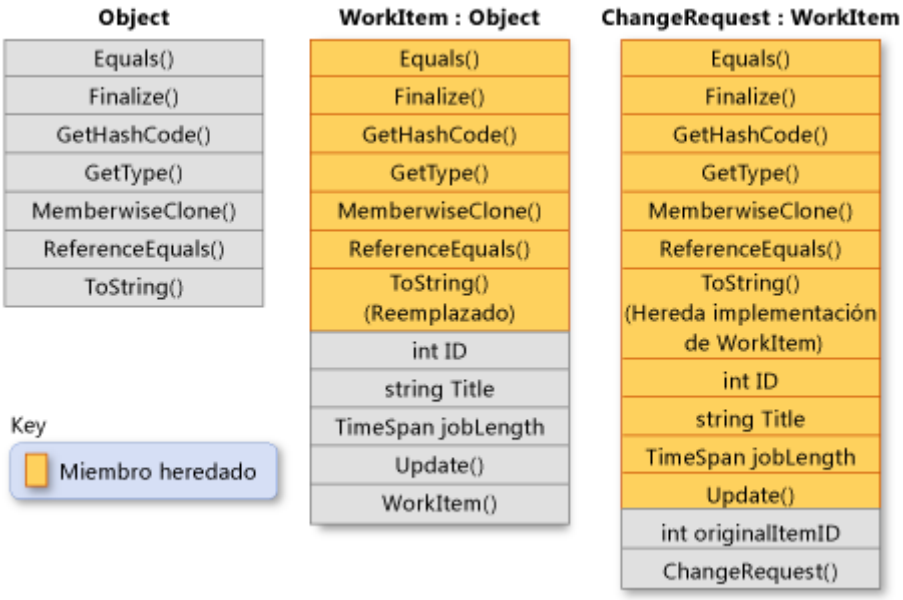

Imagen 4. Ejemplo de herencia.

- **Polimorfismo:** es una de las técnicas más poderosas de la programación orientada a objetos, permitiendo que el código de una función se escriba una vez, con un alto nivel de abstracción. El tipo de los argumentos no se conoce en el momento en que se define la función, se ajusta cuando sea necesario para corresponder a una variedad de situaciones. En general, el programador lleva a cabo este ajuste enviando más mensajes al receptor del método. A menudo estos mensajes subsecuentes no se asocian con la clase en el nivel del método polimórfico, sino que más bien son métodos virtuales definidos en las clases más bajas.

```
public class ClaseBase
{
    public virtual void Algo() { }
    public virtual int miPropiedad
    {
        get { return 0; }
    }
}
public class ClaseDerivada : ClaseBase
{
    public override void Algo() { }
    public override int miPropiedad
    {
        get { return 0; }
    }
}
```

Imagen 5. Ejemplo de polimorfismo.

2.2 Modelado y análisis conceptual.

Análogamente a la construcción de un edificio (los arquitectos realizan multitud de planos antes de empezar) el desarrollo de un modelo de un sistema software de calidad industrial antes de su construcción o renovación es tan necesario como obtener un plano para construir un rascacielos. Ello es esencial para facilitar la comunicación entre los diversos miembros de los equipos de desarrollo. El uso de un modelo proporciona varias ventajas:

- Facilita la comunicación de forma no ambigua.
- Facilita la comprensión de sistemas complejos.
- Es una solución general y reutilizable.
- Es más estable que una descripción textual.
- Es expresivo y puede ayudar a resolver problemas similares.
- Se pueden utilizar herramientas automáticas para verificar las propiedades del modelo.
- Se puede razonar acerca de él y extraer conocimiento.

En definitiva un modelo es una especificación detallada del problema del mundo real al que nos enfrentamos (orientado al problema) y está desprovisto de consideraciones de diseño e implementación.

2.2.1 Métodos de modelado orientado a objetos.

Los métodos de modelado OO indican cómo capturar y representar el conocimiento del problema desde una perspectiva orientada a objetos. Los métodos deben proporcionar un modelo de objetos bien definido, es decir, conceptos fundamentales del modelado OO y su semántica asociada (clases, relaciones entre clases, interacción entre objetos, métodos, atributos, comportamiento. . .) Dichos métodos también deben aportar una notación de los conceptos incluidos en el Modelo de Objetos, y unas guías para identificar, especificar y verificar conceptos del modelado OO. Esto hace que surja la necesidad de utilizar una notación estándar.

Existen diferentes lenguajes para modelar lo que será una aplicación orientada a objetos entre los que se encuentran OMT [7], UML [1], etc. No obstante, el que predomina hoy en día es UML (Unified Modeling Language, lenguaje de modelado unificado). UML es un lenguaje de propósito general para el modelado orientado a objetos. Presumiblemente, UML será el lenguaje de modelado orientado a objetos estándar predominante los próximos años debido a que hay una gran participación de metodólogos influyentes, participan importantes empresas (Rational Software, Digital Equipment, Hewlett-Packard, Microsoft, Oracle Corp., IBM, etc.), y ha sido aceptado como notación estándar por el OMG [6] (Object Management Group).
UML proporciona distintos diagramas útiles entre los que se encuentran los siguientes:

- **Diagrama de casos de uso**. Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de los usuarios. Los casos de uso son descripciones de la funcionalidad del futuro sistema independientes de la implementación y sirven para captar los requisitos funcionales de un sistema software.
  Estos diagramas son fácilmente comprensibles por clientes/usuarios y representan los requisitos funcionales. Estos diagramas se utilizan como base para un desarrollo iterativo e incremental, y además, son incorporados en la mayor parte de los métodos de desarrollo OO de segunda generación.

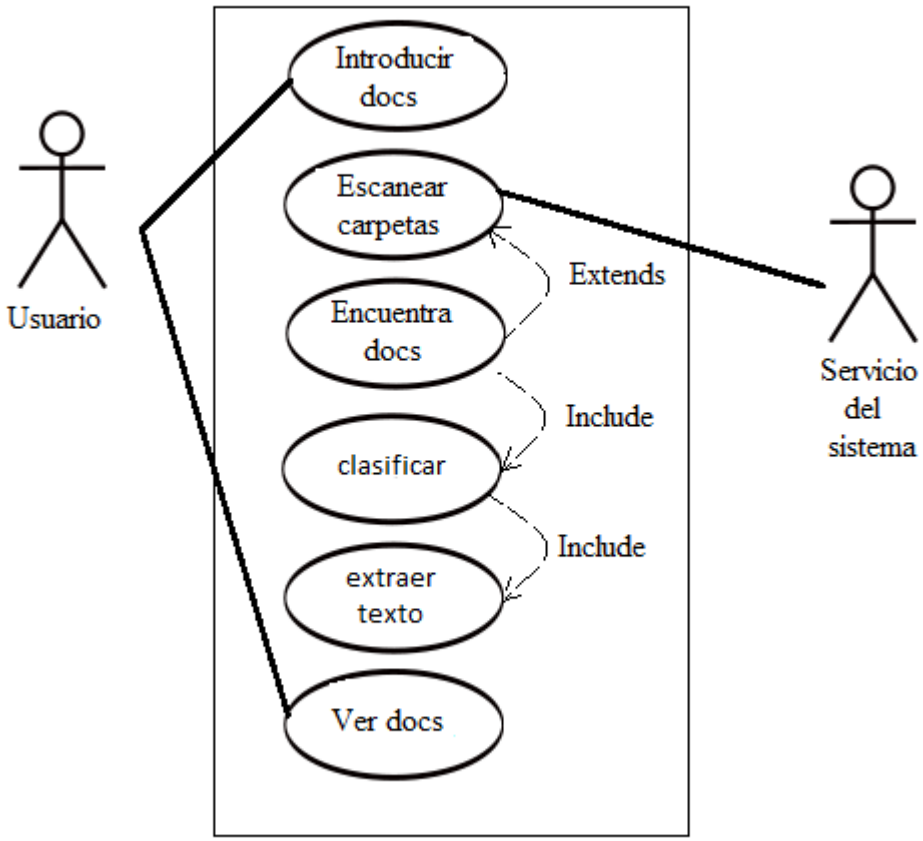

Imagen 6. Diagrama casos de uso

- **Diagrama de clases:** (incluyendo diagrama de objetos). Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. El diagrama de clases se ve con más detalle a continuación dentro de este mismo punto 2.2.1.

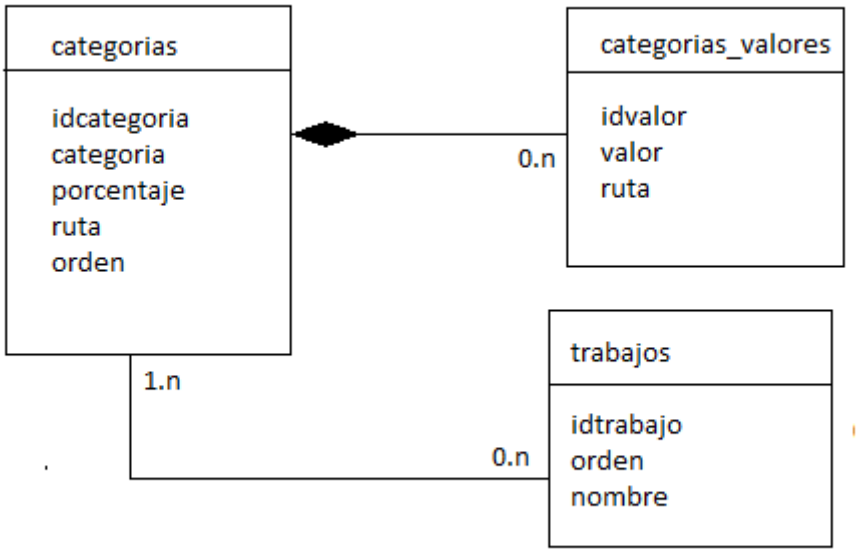

Imagen 7. Parte de un diagrama de clases

- **Diagramas de comportamiento.** El modelo de comportamiento está formado por:
  - **Diagrama de estados:** describen los ciclos de vida de los objetos y sirven para modelar la decisión de un objeto sobre qué hacer cuando recibe un mensaje. Esta decisión depende de su estado (valor de sus atributos). Es importante detectar los estados relevantes para el objeto y los eventos que afectan a dichos estados. Un diagrama de estados relaciona eventos y estados.

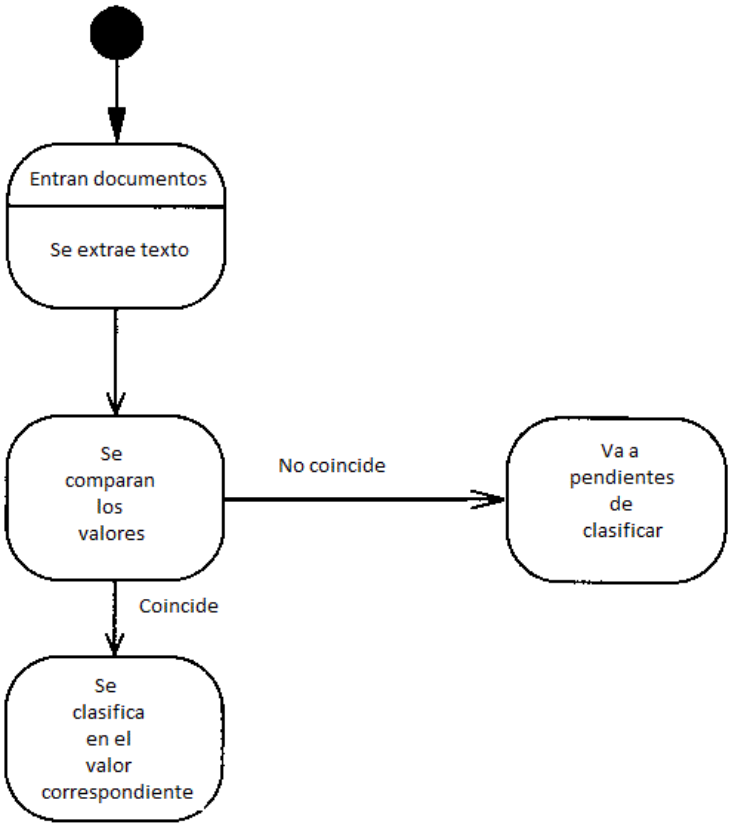

Imagen 8. Ejemplo diagrama de estados.

- **Diagrama de actividad:** describen cómo se coordinan las actividades y almacenan dependencias entre actividades, tales como qué tareas pueden darse en paralelo, qué tiene que estar terminado antes de que otra actividad empiece, etc.
- **Diagramas de interacción:** reflejan la comunicación dentro del sistema. En estos diagramas los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de interacción muestran cómo se comunican los objetos para realizar un servicio y son los siguientes:
  - **Diagrama de secuencia:** sirven para identificar la comunicación dentro del sistema y las operaciones de las clases. En ellos se muestra la secuencia de mensajes entre objetos durante un escenario concreto.

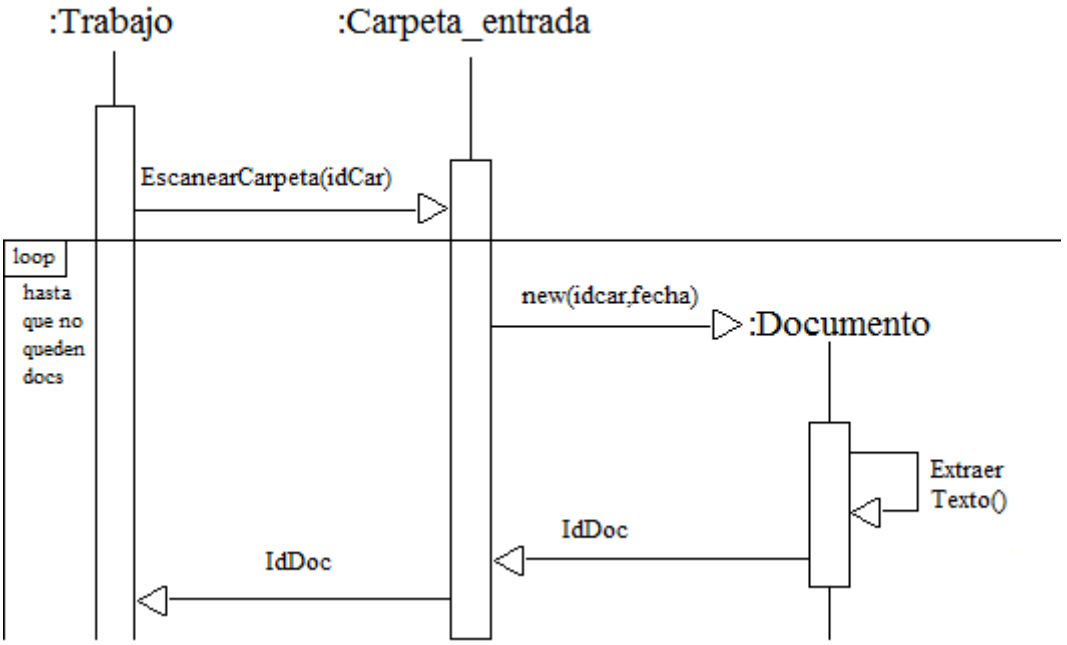

Imagen 9. Parte de diagrama de secuencia

- **Diagrama de colaboración:** muestran una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los diagramas de secuencia, los diagramas de colaboración muestran las relaciones entre los roles de los objetos.

El diseñador deberá usar unos diagramas y descartar otros en función de la metodología empleada. Hay algunos que son imprescindibles como es el diagrama de clases, que pasa a enunciarse a continuación.

Diagrama de clases

El diagrama de clases es el diagrama principal para el modelado y diseño OO. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales (asociación, agregación y composición) y de herencia. La definición de clase incluye definiciones para atributos y operaciones. Cada clase se representa gráficamente como un rectángulo con tres compartimentos (nombre de la clase, atributos de la clase y operaciones de la clase).

Los atributos y los métodos, pueden ser privados, protegidos o públicos.

Relaciones entre clases.

Las clases no actúan como elementos aislados sino que casi siempre se relacionan. Para poder entender cómo se pueden interrelacionar dos o más clases necesitamos definir el concepto de multiplicidad (también conocido como cardinalidad) que especifica el número de instancias de una clase que pueden estar relacionadas con una ´única instancia de una clase asociada.

1	Uno y sólo uno
0..1	Cero o uno
X..Y	Desde X hasta Y (números naturales)
*	Cero o muchos
0..*	Cero o muchos
1..*	Uno o muchos

Imagen 10. Multiplicidad.

La multiplicidad mínima puede establecer restricciones de existencia para los objetos de las clases asociadas, es decir puede ser mayor que 1. Una vez visto el concepto de multiplicidad las formas de relacionarse las clases son:

- **Asociación:** permite asociar objetos entre sí(En la imagen 7 se pueden apreciar dos asociaciones, entre categorías y categorías valores y categorías y trabajos).
- **Agregación:** se trata de un caso particular de asociación. Es una relación parte de en la cual los objetos que representan los componentes de algo, se asocian a un objeto que representa el ensamblaje completo. Es una forma fuertemente acoplada de asociación con más semántica. Algunas propiedades son: transitividad, antisimetría, propagación.

Imagen 11. Ejemplo de agregación.

- **Composición:** es un tipo especial de agregación, que impone más restricción, i.e. el todo posee fuertemente a sus partes. Las partes no tienen sentido por sí solas, ´únicamente en el contexto del todo.

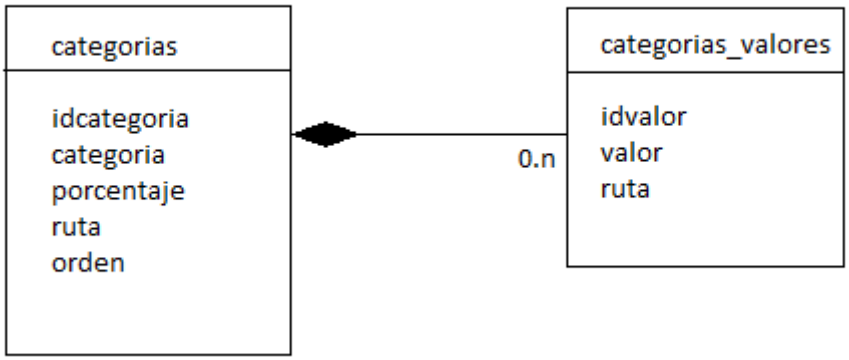

Imagen 12. Ejemplo de composición.

- **Generalización/especialización:** cuando se detecta que entre distintas entidades definidas en el esquema existe una relación de inclusión (esto es, que todas las ocurrencias de una entidad son a su vez ocurrencia de otra más general), este hecho se expresa por medio de la generalización/especialización. Esto significa que la entidad más general se especializa en una o varias entidades especializadas o subclases, o dicho a la inversa, que una o varias entidades se generalizan en una clase general o superclase. Este proceso se puede repetir a distintos niveles, siendo posible que una entidad tenga más de una superclase, siempre que la clase más general del conjunto sea única. La clase más general será además la única que tenga identificador. Todas las subclases de una clase tienen, además de sus atributos propios, todos los atributos de sus superclases (en cualquier nivel), aunque no se representan en el diagrama. La especialización de una entidad en varias subclases puede ser total (T) con lo que todas sus ocurrencias deben participar en alguna subclase, o parcial (P) en caso contrario. También tendrá la propiedad de ser solapada (S) si una ocurrencia de la entidad puede pertenecer a distintas subclases a la vez, o disjunta (D) en caso contrario. En los lenguajes orientados a objetos la

generalización/especialización se implementa como herencia y además permite gestionar la complejidad mediante un ordenamiento taxonómico. En este proceso se usan los mecanismos de abstracción de generalización y/o especialización. La generalización consiste en factorizar los elementos comunes de un conjunto de clases en una clase más general.

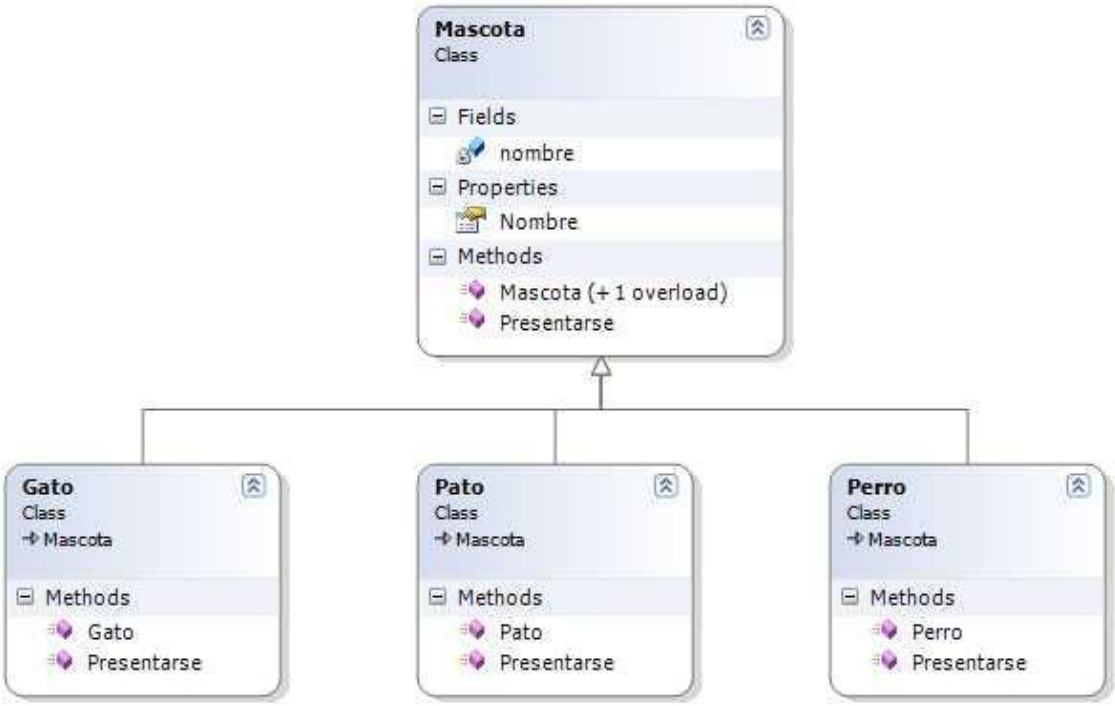

Imagen 13. Ejemplo de generalización y especificación.

2.3 Diseño conceptual.

Una vez se ha analizado el problema, es necesario decidir la estrategia de alto nivel que nos ayude a construir la solución software del modelo conceptual. A esta estrategia se le llama diseño. La estrategia de diseño propuesta es la siguiente:

1. Descripción de la arquitectura del sistema:

   Identificar el tipo de arquitectura que permita resolver el problema.

   Descomponer el sistema en subsistemas.

   Elegir una estrategia de diseño para la implementación de la gestión de datos, soporte a la interfaz de usuario y gestión de tareas.

2. Diseño de objetos:

   Representar las Clases, Asociaciones, Agregaciones y relaciones de Herencia.

   Diseñar las estructuras de datos internas para la representación y almacenamiento de los objetos y atributos.

   Diseñar los métodos básicos (lectura y escritura, inserción y borrado de objetos).

3. Diseño de mensajes:

   Diseñar la interacción entre objetos a partir de las colaboraciones detectadas en la descripción de escenarios.

   Diseñar los algoritmos.

   El diseño debe ser un proceso iterativo e incremental que podrá llevarse a cabo concurrentemente con las tareas de análisis.

   La descripción de la Arquitectura del Sistema es la primera etapa del diseño. En esta etapa debemos dotar al sistema de una organización global en subsistemas o componentes, lo que llamamos arquitectura. En ella se definirán subsistemas (conjunto de clases o componentes funcionalmente interrelacionadas) y se presentan una serie de patrones arquitectónicos típicos en los sistemas de información: sistemas interactivos, sistemas multinivel, sistemas distribuidos, etc. Apoyándose en estos patrones, se elige una estrategia de diseño para la implementación de la gestión de datos, el soporte a la interfaz de usuario y la gestión de tareas.

   En la fase de diseño de objetos se explica cómo implementar en un lenguaje de programación en particular los modelos del dominio de la fase de análisis, cómo representar las clases, cómo implementar asociaciones y agregaciones, qué estructuras de datos internas son necesarias para la representación y almacenamiento de los objetos y atributos, cómo ajustar las relaciones de herencia (eliminando o añadiendo subclases, reorganizando el estado y comportamiento de estas), y cómo utilizar la delegación mediante agregación para implementar relaciones de herencia que no son válidas semánticamente. En esta fase se obtienen, además, los métodos básicos de lectura y escritura, y los métodos de inserción y borrado de objetos en agregaciones o asociaciones.

En la última fase de diseño de mensajes se aborda la especificación detallada de las colaboraciones detectadas en los diagramas de secuencia (o trazas de eventos) obtenidos a partir de los escenarios, para la implementación de la interacción entre objetos. Las trazas de eventos nos servirán para detectar métodos que inicialmente no habíamos encontrado y que con la descripción de la interacción entre objetos aparecen explícitamente. Se explicará por tanto cómo convertir los eventos de una traza de eventos en métodos. También se detallarán los métodos más complejos que incluyen llamadas a diversos objetos colaboradores.

2.3.1 Arquitectura de la aplicación

Durante la fase de desarrollo de la arquitectura se proporciona a la aplicación una estructura y un estilo global. El diseñador desglosa el sistema en subsistemas de tal manera que sea posible realizar más trabajo por parte de varios diseñadores trabajando independientemente en los distintos subsistemas. En todas las aplicaciones el primer paso para diseñar un sistema consiste en dividir el sistema en un pequeño número de componentes llamados subsistemas.
Cada subsistema abarca aspectos del sistema que comparten propiedades en común (funcionalidad similar, tipo de funcionalidad, ubicación física, etc.) y se identifica por los servicios que proporciona. La descomposición de sistemas en subsistemas se puede organizar como una secuencia de capas horizontales (arquitectura multicapa) construido en términos de los que tiene por debajo, y proporciona la base de la implementación de aquellos que estén por encima de él. Los objetos de cada capa pueden ser independientes (recomendado) aunque suelen haber correspondencias entre objetos de distintas capas. Existe una relación cliente/servidor entre las capas inferiores (que proporcionan servicios) y las capas superiores (que son usuarios de estos servicios). Se recomienda trabajar con arquitecturas cerradas ya que reducen las dependencias entre capas y permiten que los cambios se hagan con facilidad porque la interfaz de una de ellas sólo afecta a la capa siguiente.

2.3.2 Arquitectura de tres capas.

Una de las arquitecturas multicapa que más se está utilizando actualmente de forma extendida es la arquitectura de tres capas lógicas. En dicha arquitectura tenemos las siguientes capas:

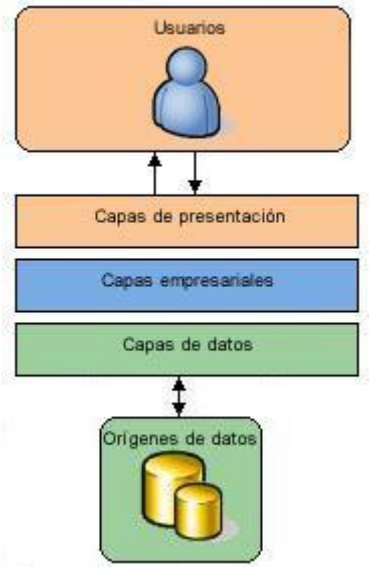

Imagen 14. Ejemplo arquitectura 3 capas

1. Nivel de presentación: Componentes software que implementan la interacción con los usuarios a través de una representación visual de la aplicación, proporcionando a los usuarios una forma de acceder y controlar los datos y los servicios de los objetos.
2. Nivel de negocio o lógica: Componentes software que implementan completamente el comportamiento de las clases del dominio, especificadas en la fase de modelado conceptual.
3. Nivel de persistencia o base de datos: Componentes software que proporcionan una serie de servicios que permiten a los objetos del dominio interactuar con su repositorio permanente asociado.

La arquitectura global de un sistema se puede seleccionar basándose en su similitud con otros sistemas anteriores. Algunos tipos de arquitecturas de sistemas son ´útiles para resolver una amplia gama de problemas como: sistemas interactivos, sistemas multinivel, sistemas distribuidos, etc.

2.4 Diseño de la base de datos.

El tratamiento automático de la información, o gestión de los datos, ha experimentado una profunda evolución desde los pioneros sistemas de gestión de ficheros hasta los actuales sistemas de gestión de bases de datos. En la actualidad las técnicas de bases de datos representan la tecnología informática disponible para la organización y gestión de grandes volúmenes de datos, ocupando un lugar relevante en todas las áreas en las que son utilizados los ordenadores: gestión, ingeniería, ciencia, medicina, etc. [3] Si realizar un buen diseño de objetos era importante, no lo es menos el realizar un buen diseño de la base de datos. Afortunadamente a partir del diagrama de clases UML, junto con las restricciones (requisitos) que en dicho diagrama no se pueden representar, puede obtenerse el esquema lógico de la base de datos

A partir de un diagrama de clases UML o bien a partir de los requisitos, es posible hacer un diagrama para implementar las tablas en el Gestor de Bases de Datos (aunque también es posible realizar la implementación de tablas directamente del modelo de clases UML). Uno de los diagramas más usados para este fin es el diagrama entidad-relación, comentado a continuación.

2.4.1 Modelo entidad-relación

El modelo entidad-relación (ER) [4] permite representar las estructuras que constituyen el contenido del sistema de información junto con restricciones de distintos tipos que limitan las ocurrencias validas de las mismas. Para ello el modelo hace uso, fundamentalmente, de tres conceptos: entidad, atributo y relación. Además, para aumentar la capacidad expresiva del modelo también se contempla la definición de objetos compuestos mediante la agregación de entidades y la definición de objetos especializados (o generalizados).

Una entidad (en UML una clase) caracteriza a un tipo de objeto, real o abstracto, del problema a modelar. Toda entidad tiene

existencia propia, se distingue del resto de las entidades, tiene nombre y posee atributos definidos en un dominio determinado. Una entidad es todo aquello de lo que se desea almacenar información. En el diagrama ER las entidades se representan gráficamente mediante rectángulos.
Una relación es una asociación o relación maten ática entre varias entidades. Las relaciones también se nombran y se representan en el diagrama ER mediante flechas y rombos.
Cada entidad interviene en una relación con una determinada cardinalidad. La cardinalidad (número de instancias o elementos de una entidad que pueden asociarse a un elemento de la otra entidad relacionada) se representa mediante una pareja de datos, en minúsculas, de la forma "cardinalidad mínima, cardinalidad máxima", asociada a cada uno de las entidades que intervienen en la relación.

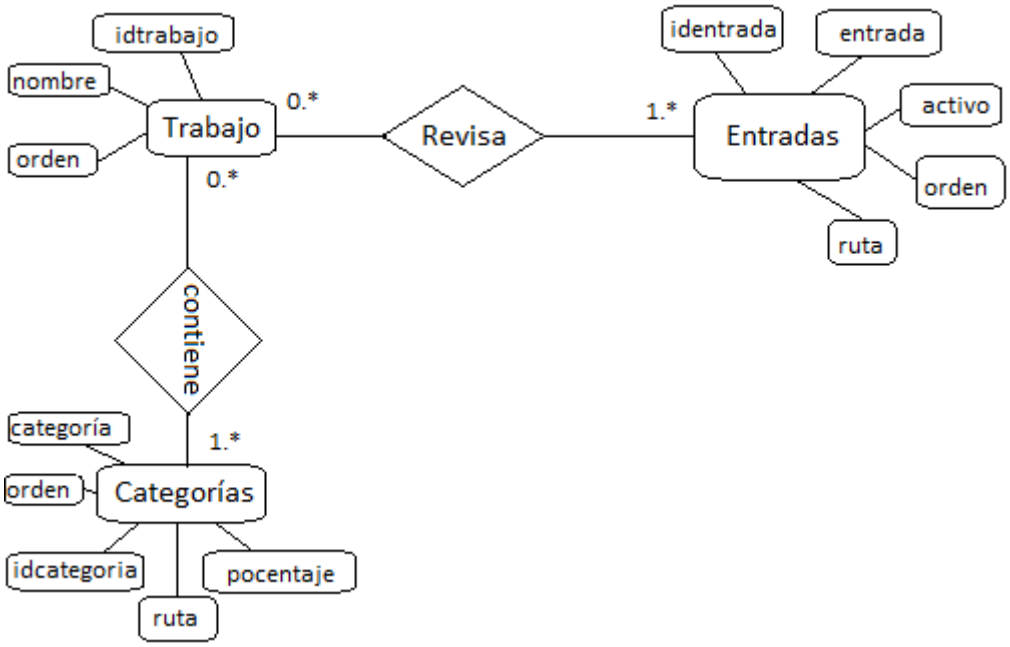

Imagen 15. Fragmento diagrama entidad-relación.

2.5 Interfaz gráfica de usuario.

La interfaz gráfica de usuario (IGU o GUI) es un punto importante de cualquier aplicación visual. Una aplicación contiene varios formularios entre los que están el formulario principal, otros formularios como cuadros de diálogo con el

usuario, ventanas secundarias, formularios de entrada de datos, etc. Una interfaz de usuario pobre produce reducción de productividad, tiempos de aprendizaje inaceptables, niveles de errores que producen frustración y, como consecuencia de todo esto, rechazo del sistema y vida útil limitada [8].

El objetivo de la interacción hombre máquina es desarrollar o mejorar la seguridad, utilidad, efectividad, eficiencia y sobre todo la usabilidad de los sistemas interactivos. Un sistema es usable si es fácil de aprender y de utilizar. La usabilidad es mucho más que la selección de colores o tipos de letra; incluye diseño de los diálogos, enlace cognitivo entre usuario y sistema.

A la hora de diseñar la interfaz gráfica de usuario, el diseñador puede basarse en su propia experiencia, o bien en la de otros diseñadores que se recogen por medio de compendios de recomendaciones relevantes para la construcción de una interfaz con ´éxito (guías, estándares, reglas de oro, etc.). Éstas son algunas de las reglas recomendadas en el diseño de interfaces de usuario:

Consistencia, en términos de sintaxis, terminología, acciones y distribución de información.

Métodos abreviados (shortcuts), proporcionando facilidades de interacción al usuario experto.

Información de feedback, proporcionando respuestas del sistema (de modesta a sustancial).

Diseño organizado y agrupado, con las secuencias de acciones claramente estructuradas desde principio a fin (cronología intuitiva).

Prevención y gestión sencilla de errores, evitando que el usuario pueda cometer errores graves (y si se cometen, detectarlos).

Permitir deshacer acciones, reduciendo la ansiedad del usuario para familiarizarse con nuevas opciones.

Control del usuario, para que el usuario sea el agente ejecutor de acciones y no un simple receptor.

Minimizar la memorización a corto plazo, con diseños sencillos e intuitivos (realidad y metáforas).

A la hora de diseñar la interfaz de usuario, si se hace uso de estas reglas se consigue que los usuarios se encuentren más cómodos con el sistema. Cuando los usuarios hacen muchas cosas a la vez,

no necesitan memorizar datos que van a utilizar unos instantes después, por ejemplo es más fácil seleccionar un elemento de una lista que acordarse del mismo.
En el diseño de la interfaz gráfica de usuario se siguen una serie de pasos entre los que se encuentran:

I. Diseño de un prototipo
II. Evaluación del mismo por los usuarios
III. Producir un prototipo ejecutable
IV. Evaluarlo nuevamente con los usuarios

Repetiremos este proceso iterativamente hasta llegar al prototipo final.

2.6 Diseño del prototipo.

Un prototipo es una primera versión de un producto, en el que se han incorporado sólo algunas características del sistema final, o no se ha realizado completamente. Los prototipos ayudan a los clientes a establecer claramente los requisitos y ayudan a los desarrolladores a mejorar sus productos. Hay dos clases de prototipos: el prototipo vertical que desarrolla completamente alguna de las facetas de la aplicación y el prototipo horizontal que desarrolla en parte todas las facetas. El prototipo horizontal es una versión de la aplicación solamente a nivel gráfico y sin funcionalidad. A partir de esa versión se ira refinando hasta llegar al estado deseado por el cliente.

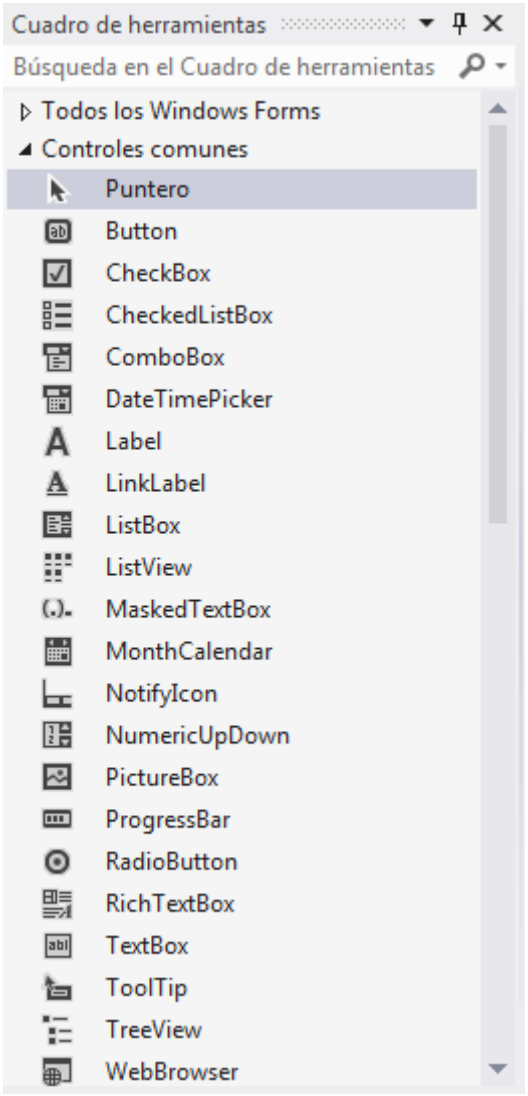

Imagen 16. Cuadro de herramientas de visual studio 2012

2.7 OCR.

El Reconocimiento Óptico de Caracteres (ROC), o generalmente como reconocimiento de caracteres, es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos, así podremos interactuar con estos mediante un programa de edición de texto o similar. Con frecuencia es abreviado en textos escritos en el idioma español, utilizando el acrónimo a partir del inglés OCR.

En los últimos años la digitalización de la información (textos, imágenes, sonido, etc.) ha devenido un punto de interés para la sociedad. En el caso concreto de los textos, existen y se generan continuamente grandes cantidades de información escrita, tipográfica o manuscrita en todo tipo de soportes. En este contexto, poder automatizar la introducción de caracteres evitando la entrada por teclado, implica un importante ahorro de recursos humanos y un aumento de la productividad, al mismo tiempo que se mantiene, o hasta se mejora, la calidad de muchos servicios. Para más información [17].

En nuestro caso, podemos usar esto para extraer el texto de cualquier documento y conseguir clasificarlo automáticamente lo que ahorra una gran cantidad de tiempo.

Imagen 17. Ejemplo de extracción de texto de una factura

Al habar de OCR nos encontramos con varios tipos.

2.7.1 Reconocimiento de texto manuscrito

Las dificultades que podemos encontrar a la hora de reconocer un texto tipografiado, no se pueden comparar con las que aparecen cuando queremos reconocer un texto manuscrito.

El reconocimiento de un texto manuscrito continúa siendo un desafío. Aunque el texto se compone básicamente de caracteres individuales, la mayoría de algoritmos OCR no consiguen buenos resultados, ya que la segmentación de texto continuo es un procedimiento complejo.

En el caso de reconocimiento de escritura manuscrita a la hora de corrección de exámenes, existe la posibilidad, añadiendo un listado de léxico (nombres y apellidos) de acercarse al 100% de acierto. A través de las casillas de respuesta ICR se pueden reconocer palabras, como nombres de países, nombres de regiones, marcas comerciales, en resumen, todo aquello que pueda ser integrado en una lista de palabras este puede ir aumentándose según necesidades.

Por otro lado, se puede llegar a comprender una frase cuando la hemos terminado de leer. Esto implica una operación de niveles morfológicos, léxico y sintáctico que se consigue mediante el reconocimiento del habla continua. Para llevar a cabo esa metodología, se utilizan algoritmos robustos que utilizan una segmentación previa, debido a que se obtiene automáticamente con la descodificación.

2.7.2 Reconocimiento de matrículas

Una de las aplicaciones son los radares. Estos deben ser capaces de localizar una matrícula de un vehículo con condiciones de iluminación, perspectiva y entorno variables.

En la etapa de segmentación, se buscan texturas similares a la de una matrícula y se aísla el área rectangular que forma la matrícula.

Finalmente, se aplica un proceso de clasificación múltiple sobre el conjunto de píxeles pertenecientes a la matrícula, proporcionando una cadena de caracteres que se tienen que ajustar a un modelo

conocido: el formato de una matrícula. Si aparece algún error, es corregido.

2.7.3 Indexación en bases de datos

Con el gran aumento de información publicada que ha tenido lugar en los últimos años, cada vez son más los métodos que se utilizan para organizar todo este material almacenado en bases de datos. Uno de estos contenidos son las imágenes. Una de las formas más corrientes de buscar imágenes es a partir de metadatos introducidos manualmente por los usuarios. Actualmente han aparecido buscadores que proporcionan la posibilidad de buscar imágenes mediante el texto que aparecen en ellas, como el buscador DIRS(Document Image Retrieval System) que, mediante un algoritmo de Reconocimiento Óptico de Caracteres, extrae el texto que aparece en la imagen y lo utiliza como metadato que podrá ser utilizado en las búsquedas. Esta tecnología proporciona una posibilidad en la búsqueda de imágenes y demuestra que el OCR aún puede dar mucho de sí.

2.7.4 Reconocimiento de datos estructurados con OCR Zonal

Se usa para digitalizar de forma masiva grandes cantidades de documentos estructurados o semiestructurados (facturas, nóminas, albaranes, pólizas, justificantes bancarios, etc.), catalogando automáticamente los documentos con los metadatos obtenidos y archivándolos en formato digital de forma indexada para facilitar su posterior búsqueda. Tiene el inconveniente de que es necesario diseñar previamente las plantillas, pero con una buena configuración se ahorra mucho tiempo en el proceso de digitalización.

2.8 Conclusión.

La metodología orientada a objetos es una de las más utilizadas en la actualidad para el proceso que se conoce como ingeniería del software cuya finalidad es producir software de gran calidad. El proceso de la ingeniería del software comienza con el modelado y análisis conceptual; a continuación se realiza el diseño conceptual, se diseña la base de datos y la interfaz gráfica de usuario. En este capítulo se ha presentado el marco básico de estas etapas.

3 Especificación de requerimientos software

En este capítulo se presenta la especificación de requisitos software de la aplicación para el proyecto de gestión documental.

3.1 Introducción.

El análisis de requisitos es el primer paso del proceso de ingeniería del Software. El propósito del Análisis de Requisitos es obtener una descripción lógica del sistema. Los desafíos principales en la ejecución y cumplimiento del método son:

- Para el analista, entender con precisión lo que el usuario desea que realice el sistema objetivo de estudio.
- Para el usuario, entender con precisión las cualidades que el producto software es capaz de ofrecerle.

La clave del éxito en esta etapa del análisis del sistema es, indudablemente, lograr establecer una correcta comunicación entre usuario y analista, de forma que cada uno de ellos pueda completar los conocimientos del otro para poder generar una visión completa de los requisitos y especificaciones.

Así pues, en esta especificación de requisitos trataremos de identificar los objetivos generales de la especificación, así como el producto software a desarrollar. En nuestro caso se han seguido algunas de las directrices del estándar IEEE 830-1998 [5]. La especificación de requerimientos tiene como objetivo:

i. ayudar a los clientes a describir claramente lo que se desea obtener mediante el software
ii. ayudar a los desarrolladores a entender qué quiere exactamente el cliente
iii. servir de base para desarrollos de estándares de especificación de requerimientos software particulares para cada organización; definiendo su formato y contenido.

3.2 Propósito.

El propósito de la especificación de este capítulo es definir aquellos requerimientos que debe tener la aplicación a realizar para gestión documenta. El motivo es obtener un buen resultado final, un desarrollo software de calidad, minimizando los errores y cumpliendo con las funcionalidades establecidas.

3.3 Descripción general.

En esta sección describiremos de forma general la aplicación aportando sus funciones, así como las características del usuario que acceda a este producto. Por ´ultimo veremos las principales restricciones de nuestra aplicación.

3.3.1 Funciones del producto.

Las funciones que exponemos a continuación, serán desarrolladas de forma explícita a lo largo del siguiente punto.

- Operaciones sobre trabajos.
  - Añadir un nuevo trabajo.
  - Modificar los datos de un trabajo.
  - Eliminar un trabajo.
  - Subir y bajar el orden en la lista de trabajos.
  - Añadir una carpeta de entrada al trabajo.
  - Añadir una categoría al trabajo.
- Operaciones sobre carpetas de entrada.
  - Añadir una nueva carpeta de entrada.
  - Modificar los datos de una carpeta de entrada.
  - Eliminar una carpeta de entrada.
  - Subir y bajar el orden en la lista de carpetas de entrada.
- Operaciones sobre categorías.
  - Añadir una nueva categoría.
  - Modificar los datos de una categoría.
  - Eliminar una categoría.
  - Añadir un valor a la categoría.

    - Subir y bajar el orden en la lista de categorías.
- Operaciones sobre valores.
    - Añadir un nuevo valor a una categoría.
    - Modificar los datos a un valor.
    - Eliminar un valor.
    - Añadir un valor a un documento pendiente de clasificar.
- Operaciones sobre documentos.
    - Clasificar un documento.

3.3.2 Características del usuario.

El usuario de la aplicación deberá tener unos requisitos mínimos de manejo con el ordenador, y unas pocas nociones de manejo de programas basados en ventanas gráficas con menú sobre el sistema operativo Windows.

3.4 Requisitos específicos.

En esta sección especificaremos más detalladamente las operaciones de la aplicación. Se ha elegido la organización por clases/objetos puesto que se adapta bien al tipo de problema
y sobre todo al enfoque orientado a objetos seguido en el desarrollo de la aplicación. Para representar los atributos en las tablas se ha utilizado la siguiente notación:

- Para representar caracteres alfanuméricos se ha utilizado la letra A. Así A (10) significa una cadena alfanumérica de 10 caracteres.
- Para representar caracteres numéricos se ha seguido una notación igual a la utilizada para representar los caracteres alfanuméricos pero utilizando la letra N. De esta forma N (10) representa una cadena numérica de diez dígitos.
- Para representar vectores se ha seguido una notación igual a la utilizada para representar los caracteres alfanuméricos pero utilizando la letra V. De esta forma V (10) representa un vector con un tamaño de 10.

- La representación del tipo fecha se expresa de la forma dd/mm/aaaa.

3.4.1 Clase Trabajo.

La clase Trabajo es la encargada de gestionar la revisión de carpetas de entrada y revisar a que categoría de cada valor ha de asignarse.

Atributos

Nombre	Tipo	Formato	Ejemplo
Idtrabajo	Número	N(3)	10
Orden	Número	N(3)	5
Nombre	Texto	A(50)	BuscaFact

Operaciones

Las operaciones de la clase trabajo se componen de las de mantenimiento (alta, modificar y eliminar) junto a subir y bajar el orden, añadir carpetas de entrada y categorías, escanear carpetas y clasificar documentos.

Dar de alta un nuevo trabajo.	
Entradas	Nombre, carpetas de entrada y categorías.
Proceso	Se calcula un nuevo id y se le inserta al final de la lista con el número de orden más alto.
Salidas	Se añade a la base de datos.

Modificar datos trabajo	
Entradas	Nombre, carpetas de entrada y categorías.
Proceso	Se modifican los datos anteriores.
Salidas	Se realizan los cambios en la base de datos.

Eliminar un trabajo.

Entradas	Idtrabajo.
Proceso	Se selecciona un trabajo de la lista y se presiona el botón eliminar.
Salidas	Se elimina de la base de datos.

Subir orden trabajo	
Entradas	Idtrabajo.
Proceso	Se selecciona un trabajo de la lista y se presiona el botón subir.
Salidas	Sube una posición en la lista y se modifica el orden en la base de datos

Bajar orden trabajo	
Entradas	Idtrabajo.
Proceso	Se selecciona un trabajo de la lista y se presiona el botón bajar.
Salidas	Baja una posición en la lista y se modifica el orden en la base de datos

Añadir carpeta de entrada al trabajo	
Entradas	Idtrabajo e idcarpeta.
Proceso	Se selecciona un trabajo de la lista y se presiona el botón añadir carpeta de entrada, se selecciona la carpeta y se añade.
Salidas	Se añade una carpeta de entrada al trabajo.

Añadir categoría al trabajo	
Entradas	Idtrabajo e idcategoría.
Proceso	Se selecciona un trabajo de la lista y se presiona el botón añadir categoría, se selecciona la categoría y se añade.
Salidas	Se añade una categoría al trabajo.

Escanear carpeta	
Entradas	Identrada.
Proceso	Se escanea la carpeta y se identifican los documentos que contiene.
Salidas	Se añaden los documentos a la base de datos.

Clasificar documento	
Entradas	Iddocumento e idcategoría
Proceso	Se selecciona un documento, se extrae el texto que contiene y se compara con las anticlaves y las claves de los valores de las categorías.
Salidas	Si el documento cumple el porcentaje de un valor se clasifica con ese valor. Si no cumple el porcentaje de ningún valor o cumple el porcentaje de varios valores va a documentos pendientes de clasificar.

3.4.2 Clase carpeta de entrada.

La clase carpeta de entrada es la que gestiona las carpetas que recibirán los documentos a clasificar.

Atributos

Nombre	Tipo	Formato	Ejemplo
Identrada	Número	N(3)	7
Ruta	Texto	A(50)	C:\Users\usuario\Entrada
Orden	Número	N(3)	3
Entrada	Texto	A(50)	EntradaFact
Activo	Número	N(1)	0

Operaciones

Las operaciones de la clase carpeta de entrada se componen de las de mantenimiento (alta, modificar y eliminar) junto a subir y bajar el orden.

Dar de alta una nueva carpeta de entrada.	
Entradas	Entrada, activo y ruta.
Proceso	Se calcula un nuevo id y se le inserta al final de la lista con el número de orden más alto.
Salidas	Se añade a la base de datos.

Modificar datos carpeta de entrada.	
Entradas	Entrada, activo y ruta.
Proceso	Se modifican los datos anteriores.
Salidas	Se realizan los cambios en la base de datos.

Eliminar una carpeta de entrada.	
Entradas	IdEntrada.
Proceso	Se selecciona una carpeta de entrada de la lista y se presiona el botón eliminar.
Salidas	Se elimina de la base de datos.

Subir orden carpeta de entrada	
Entradas	IdEntrada.
Proceso	Se selecciona una carpeta de la lista y se presiona el botón subir.
Salidas	Sube una posición en la lista y se modifica el orden en la base de datos

Bajar orden carpeta de entrada	
Entradas	IdEntrada.
Proceso	Se selecciona una carpeta de entrada de la lista y se presiona el botón bajar.
Salidas	Baja una posición en la lista y se modifica el orden en la base de datos

3.4.3 Clase categoría.

La clase categoría se encarga de separar los documentos en distintos tipos y es la que contiene los valores.

Atributos

Nombre	Tipo	Formato	Ejemplo
IdCategoria	Número	N(3)	9
Ruta	Texto	A(50)	C:\Users\usuario\Factura
Orden	Número	N(3)	8
Categoria	Texto	A(50)	Factura
Porcentaje	Número	N(3)	75

Operaciones

Las operaciones de la clase categoría se componen de las de mantenimiento (alta, modificar y eliminar) junto a subir y bajar el orden y añadir valor a la categoría.

Dar de alta una nueva categoría.	
Entradas	Categoría, porcentaje y ruta.
Proceso	Se calcula un nuevo id y se le inserta al final de la lista con el número de orden más alto.
Salidas	Se añade a la base de datos.

Modificar datos categoría.	
Entradas	Categoría, porcentaje y ruta.
Proceso	Se modifican los datos anteriores.
Salidas	Se realizan los cambios en la base de datos.

Eliminar una categoría.	
Entradas	IdCategoria.
Proceso	Se selecciona una categoría de la lista y se

	presiona el botón eliminar.
Salidas	Se elimina de la base de datos.

Subir orden categoría.	
Entradas	IdCategoria.
Proceso	Se selecciona una categoría de la lista y se presiona el botón subir.
Salidas	Sube una posición en la lista y se modifica el orden en la base de datos

Bajar orden categoría.	
Entradas	IdCategoria.
Proceso	Se selecciona una categoría de la lista y se presiona el botón bajar.
Salidas	Baja una posición en la lista y se modifica el orden en la base de datos

Añadir valor a la categoría.	
Entradas	IdCategoria e idValor.
Proceso	Se selecciona una categoría de la lista y se presiona el botón añadir valor, se selecciona el valor y se añade.
Salidas	Se añade un valor a la categoría.

3.4.4 Clase valor.

La clase valor es la que se encarga de almacenar los documentos.

Atributos

Nombre	Tipo	Formato	Ejemplo
IdValor	Número	N(3)	21
Ruta	Texto	A(50)	C:\Users\usuario\Factura\Consum
Claves	Vector	V(4)	Consum, Facturas, CIF

Anticlaves	Vector	V(4)	Proveedor
Valor	Texto	A(50)	Consum

Operaciones

Las operaciones de la clase valor se componen de las de mantenimiento (alta, modificar y eliminar) junto a añadir valor a un documento pendiente.

Dar de alta un nuevo valor.	
Entradas	Valor y ruta.
Proceso	Se calcula un nuevo id y se le inserta en la categoría seleccionada.
Salidas	Se añade a la base de datos.

Modificar datos valor.	
Entradas	Valor y ruta.
Proceso	Se modifican los datos anteriores.
Salidas	Se realizan los cambios en la base de datos.

Eliminar un valor.	
Entradas	IdValor.
Proceso	Se selecciona una valor de la lista y se presiona el botón eliminar.
Salidas	Se elimina de la categoría seleccionada.

Añadir valor a un documento pendiente.	
Entradas	IdValor.
Proceso	Se selecciona un documento pendiente de clasificar y se presiona el botón añadir valor, se selecciona el valor y se añade.
Salidas	Se añade un valor al documento pendiente.

3.4.5 Clase documento.

La clase documento es la que se encarga de gestionar los documentos.

Atributos

Nombre	Tipo	Formato	Ejemplo
IdValor	Número	N(3)	21
Ruta	Texto	A(50)	C:\Users\usuario\Factura\Consum
Valor	Texto	A(50)	Consum

Operaciones

Las operaciones de la clase valor se componen de las de mantenimiento (alta, modificar y eliminar) junto a añadir valor a un documento pendiente.

Clasificar un documento pendiente.	
Entradas	IdDocumento.
Proceso	Se selecciona un valor y se presiona el botón OK.
Salidas	El documento se clasifica.

3.5 Conclusión.

En este capítulo se ha presentado la especificación de requisitos software (ERS) de la aplicación de gestión documental llevada a cabo en este proyecto final de carrera. La especificación de un producto software ayuda a los desarrolladores a expresar lo que desean que haga el software. Por lo tanto, es muy importante haber realizado una buena especificación porque proporciona varios beneficios específicos como:

- Reducir el esfuerzo de desarrollo. La revisión cuidadosa de los requisitos puede mostrar omisiones, malentendidos e incongruencias al principio del ciclo de desarrollo cuando estos problemas son más fáciles de corregir.
- Proporcionar una buena base para la estimación de costes y planificación. La descripción del producto a ser desarrollado dado en la ERS sirve para estimar el coste temporal del proyecto y para estimar el coste económico.
- Servir de punto de referencia para procesos de verificación y validación. Las empresas pueden desarrollar sus planes de verificación y validación de una manera mucho más productiva partiendo de una buena ERS.
- Facilitar la transferencia de productos software. La ERS hace más fácil transferir el producto software a nuevos usuarios o nuevas máquinas.
- Servir de base para posibles mejoras.

En definitiva, la buena especificación nos habrá servido para dar respuesta a preguntas como: ¿Qué es lo que el software debe hacer? ¿Cómo interactúa con los usuarios y con el sistema Hardware? ¿Hay alguna restricción en cuento a estándares a seguir, lenguaje de implementación, entorno de funcionamiento, etc.?

4 Desarrollo de la aplicación.

En este capítulo se exponen los pasos seguidos para desarrollar la aplicación. Concretamente, se podrá ver cómo se han aplicado los conocimientos teóricos para abordar y solucionar el problema expuesto en el Capítulo 1. A lo largo del capítulo se sigue un orden natural de desarrollo. Primero se verá la planificación, a continuación el diseño de la aplicación, para seguir con la codificación, las pruebas, instalación y finalmente la puesta en marcha como un caso práctico de uso.

4.1 Planificación.

Por tratarse del desarrollo de un sistema de información, las fases de desarrollo coinciden plenamente con las etapas de desarrollo de software. Concretamente, estas fases son:

- Análisis y especificación de los requisitos. En esta fase se realiza un estudio previo se identifican los requisitos de usuario y los requisitos de la interfaz. Duración: 5 días.
- Modelado conceptual. En esta fase se construye el modelo de objetos, se identifican las clases y atributos. Por último se obtiene la especificación de métodos y atributos de las clases. Duración: 14 días.
- Diseño. En la fase de diseño se obtiene la arquitectura en tres niveles (presentación / lógica / persistencia) de la aplicación. Se realiza el diseño y definición del modelo relacional (BD) para el almacenamiento persistente de la información. Duración: 20 días.
- Codificación. En esta fase se realiza la implementación de la aplicación, organización en unidades y codificación de las clases usando un lenguaje de programación orientado a objetos, en este caso c#. Duración: 90 días.

- Pruebas: En la fase de pruebas se realizan las pruebas del sistema. Duración: 14 días.
- Realización de esta memoria: Una vez finalizadas las fases de desarrollo se procedió a realizar el presente documento. En él se pretende mostrar de manera precisa y concisa las pautas seguidas a lo largo de todo el desarrollo del proyecto. Duración: 40 días.

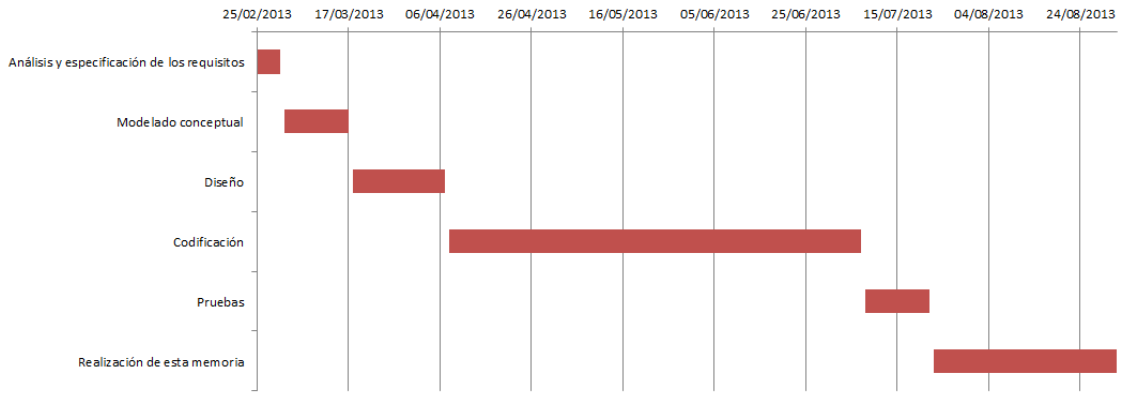

Imagen 18. Diagrama de gannt del proyecto

A continuación en los puntos siguientes del capítulo se explica cada fase con detenimiento.

4.2 Diseño de la aplicación.

El proceso de diseño ha consistido en proponer una solución al problema pensando lo que necesitaría un software de este tipo. Primero se capturan los requisitos. Una vez capturados se puede empezar a construir el prototipo horizontal. Una vez finalizada la fase de especificación de requisitos, se pasa a la fase de diseño del sistema, creando a partir de estos requisitos, el modelo de la aplicación.

4.2.1 Análisis y especificación de requisitos.

El propósito del Análisis de Requisitos (como se vio en el Capítulo 3) es obtener una descripción lógica del sistema. En mi

caso me puse en el caso de que fuera un cliente o un usuario y pensar lo que necesitaba mi software.

4.2.2 Construcción del prototipo.

En nuestro caso como no había ninguna restricción ni preferencia sobre el entorno visual de programación a utilizar, hemos optado por utilizar la herramienta RAD visual studio 2012 para el desarrollo del prototipo y de la aplicación.

Tras haber realizado la especificación de requisitos se empezó a construir el prototipo horizontal. Para ello se crearon los formularios que permiten gestionar la información de los requisitos iniciales. Para realizar los formularios se parte de los requisitos iniciales, y se pregunta la información concreta y en detalle de cada requisito.

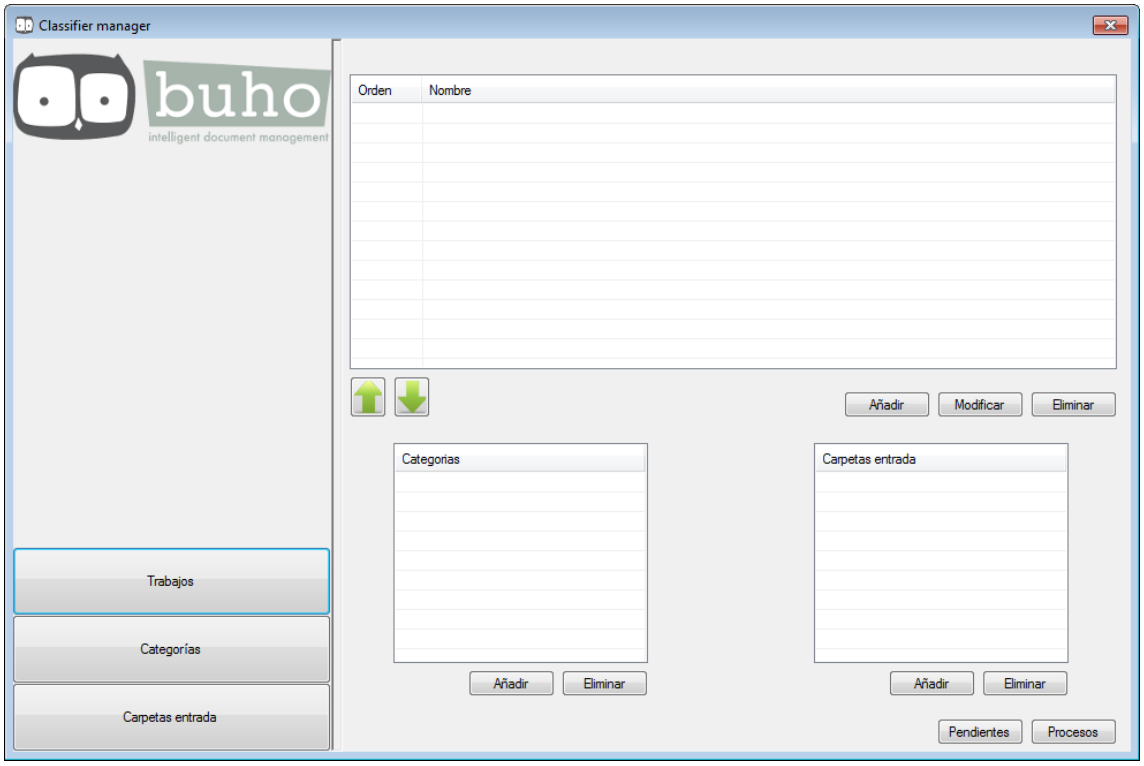

Imagen 19. Prototipo final de la aplicación.

Como se puede apreciar en la imagen para realizar el prototipo se utilizó un form de visual studio y se aprovecharon la variedad de

componentes que ofrece para realizar un formulario (Button, listview, pictureBox, etc.).

4.2.3 Análisis y diseño orientado a objetos.

Una vez conocidos los requisitos, hay que pensar en la arquitectura de la aplicación. Para ello utilizamos UML.

Uno de los diagramas que se utilizan en UML es el diagrama de usos explicado en el capítulo 2. Este se utiliza para saber las funciones que pueden realizar algunos usuarios y el comportamiento del sistema.

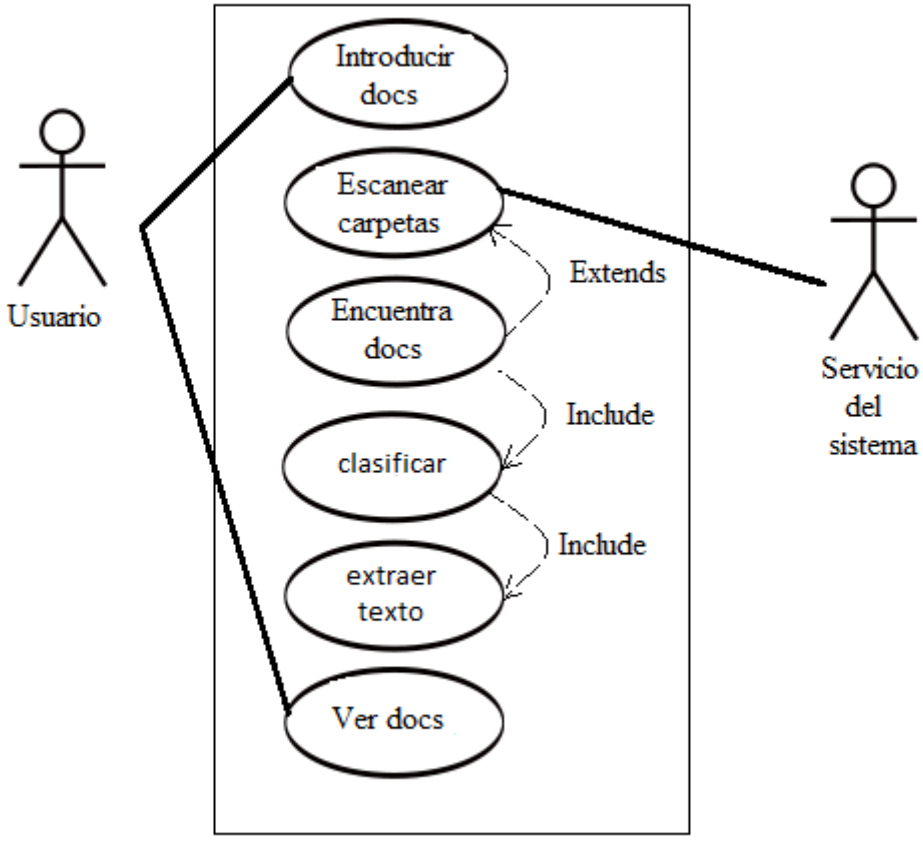

Imagen 20. Diagrama de casos de uso

Después seguimos con un diagrama que explique la estructura del programa. Un diagrama de clases, como se comentó en el Capítulo 2, presenta las clases del sistema con sus relaciones estructurales (asociación, agregación y composición) y de herencia.

También se comentó que una clase se representa gráficamente como un rectángulo con tres compartimentos (nombre de la clase, atributos de la clase y operaciones de la clase.

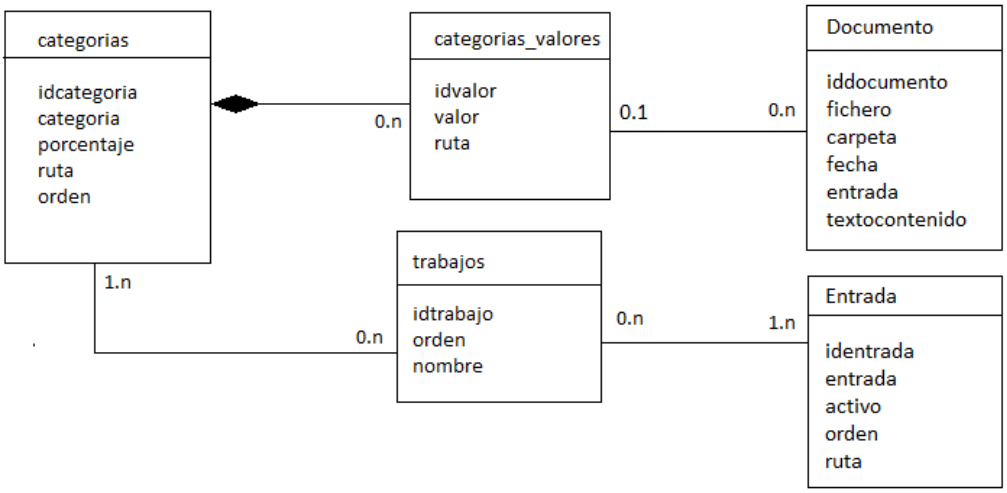

Imagen 21. Diagrama de clases UML

Y un diagrama de secuencia para los métodos más complejos, en nuestro caso para los métodos encargados de escanear carpetas y clasificar documentos.

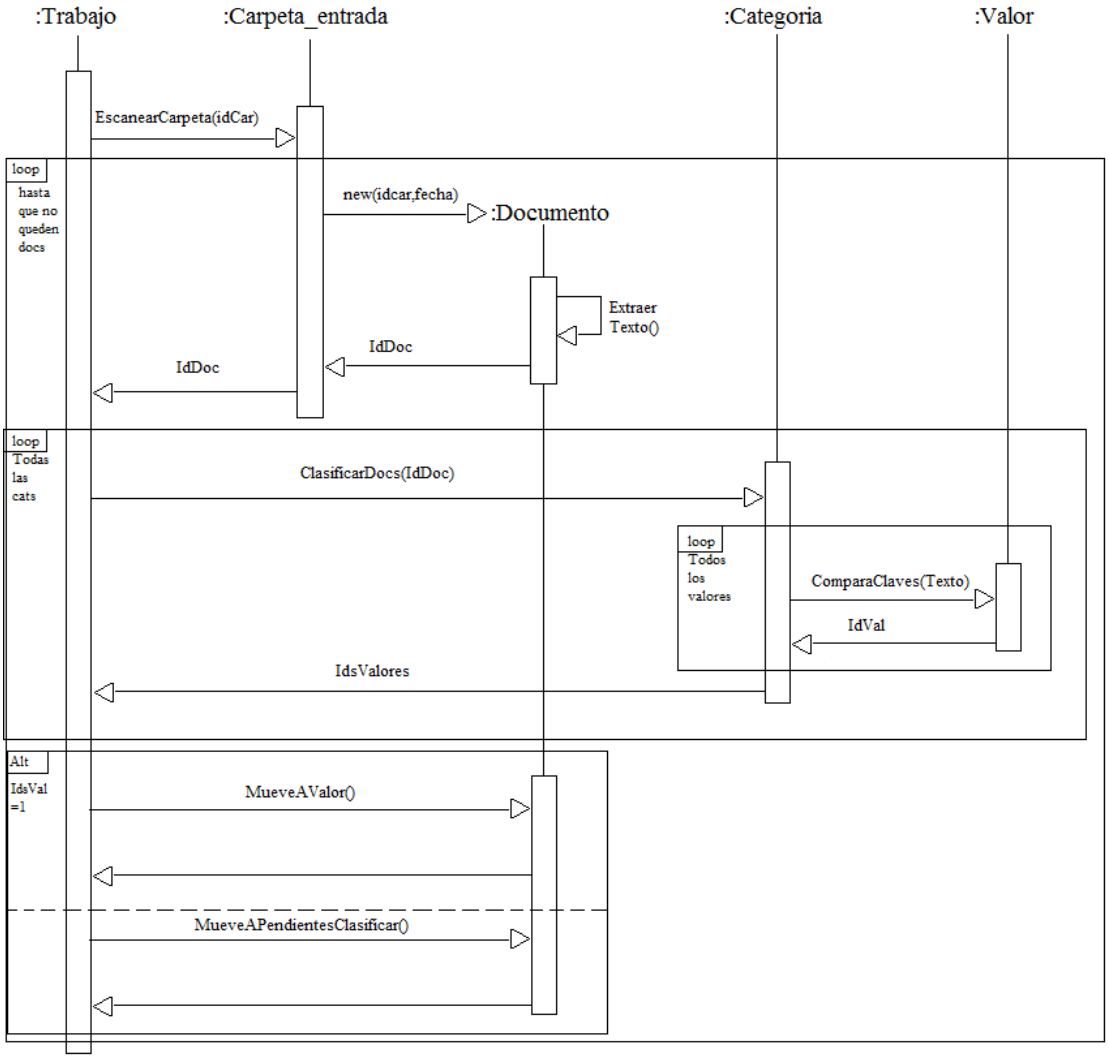

Imagen 22. Diagrama de secuencia.

4.2.4 Diagrama entidad-relación.

El modelo entidad-relación es muy similar al diagrama de clases de UML, muchas veces se complementan en los proyectos. En la siguiente imagen mostramos el diagrama de nuestra aplicación.

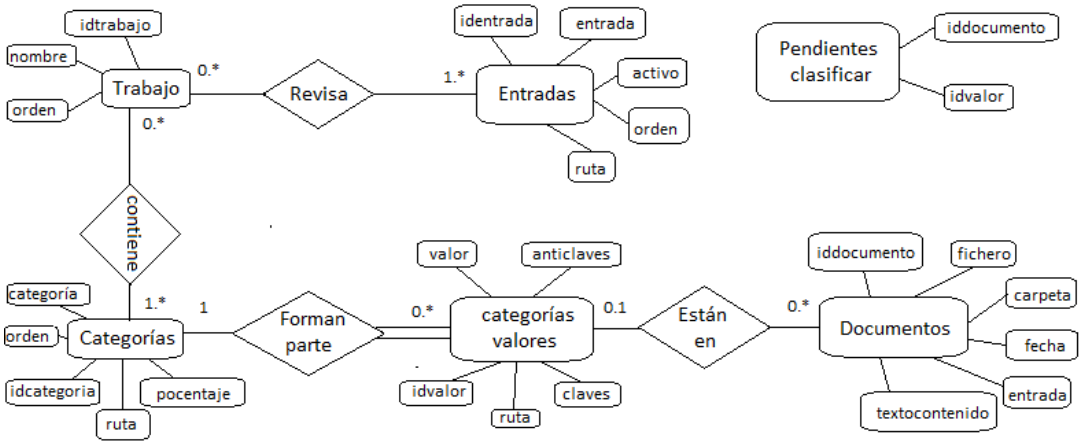

Imagen 23. Diagrama entidad-relación

4.2.5 Arquitectura de la aplicación.

La arquitectura que se ha seguido para el desarrollo de la aplicación es la arquitectura de tres capas por ser una de las más utilizadas, tal y como se comentó en el capítulo 2. En la imagen 13 del capítulo 2 puede verse el esquema de la arquitectura de tres capas.

El nivel de presentación (interfaz de usuario) estará constituido por una serie de formularios (uno de ellos el principal) que interaccionarán con el usuario y le proporcionarán la información sobre el estado de la aplicación. Estos formularios son los que se realizaron en el prototipo que ahora se reutilizan y a los que se les dota de funcionalidad. El nivel de negocio contendrá la implementación de las clases y objetos del dominio. Para el nivel de persistencia de datos se ha elegido el sistema de gestión de bases de datos PostgreSQL [13]. Un sistema de gestión de base de datos libre.

4.3 Codificación.

El siguiente paso es implementar la funcionalidad de la aplicación. Para ello se organiza en unidades y se codifican las

clases usando el lenguaje de programación orientado a objetos C#.

4.3.1 AcroPDF.

Acropdf.dll es una librería de Adobe Systems, Inc. que se usa para trabajar con documentos pdf.

En nuestra aplicación es usado de dos maneras:

- Como visor de pdf. Como en nuestra aplicación podremos guardar archivos pdf y el visor de archivos que utilizo (pictureBox) no puede mostrar archivos en pdf, he utilizado un visor que ofrece esta librería AxAcroPDF [14]. Este visor me permite ver archivos pdf en la aplicación.

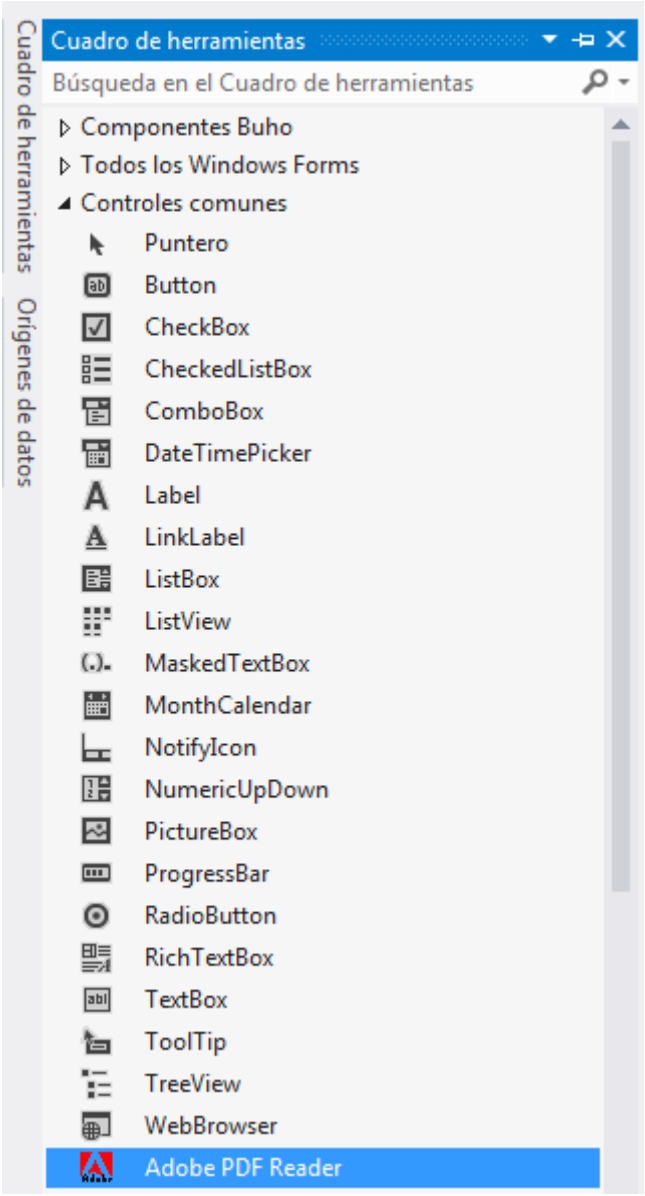

Imagen 24. PDF Reader en el cuadro de herramientas de visual studio

- Como OCR para pdf. Para ello usamos una clase incluida en la capa de negocio (pdfparser [15, 16]) que tiene un método que nos devuelve el texto de cualquier pdf.

```
PDFParser pdfParser = new PDFParser();

// extract the text
string pdf = Convert.ToString(files.Current);
String texto = pdfParser.ExtractText(pdf,"");
```

Imagen 25. Extracción de texo con pdfparser

4.3.2 MODI

Microsoft Office Document Library (MODI) [11] es el otro OCR que he utilizado. Como tener solo un OCR para pdf es poco hemos utilizado este OCR que es capaz de extraer texto de los formatos más utilizados (Excel, tiff, Word, jpg, etc.). Si juntamos estos formatos junto con pdf cubrimos la mayoría de formatos que son usados para guardar documentos de relevancia en una empresa.

Para extraer el texto simplemente usamos el objeto MODI.Document el que nos ofrece varios métodos para trabajar con distintos documentos y extraer el texto de cada uno de ello. Gracias a esto podemos extraer el texto de la mayoría de documentos y clasificarlos automáticamente.

```
//OCR Operations ...
MODI.Document md = new MODI.Document();
md.Create(Convert.ToString(files.Current));
md.OCR(MODI.MiLANGUAGES.miLANG_ENGLISH, true, true);
MODI.Image image = (MODI.Image)md.Images[0];
string texto = quitarTildesySimbolos(image.Layout.Text);
```

Imagen 26. Fragmento de código que extrae texto con MODI

4.3.3 Threads

Un hilo de ejecución (thread) es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

La creación de un nuevo hilo es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo con otra tarea.

Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar éstos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.

Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros).

El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado. Asimismo en el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

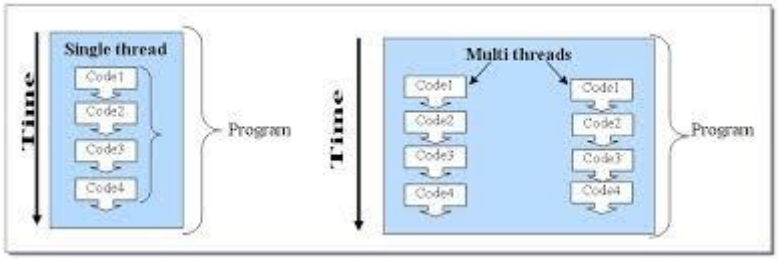

Imagen 27. Ejemplo gráfico de un thread

Como queremos automatizar el proceso de lectura de documentos de las carpetas de entrada debemos tener un proceso que continuamente mire las carpetas de entrada y clasifique los documentos. Para esta tarea fueron creados los trabajos, pero para que la parte gráfica no se quede bloqueada mientras se recorren las carpetas usamos los threads.

En nuestro programa tenemos una ventana que se encarga de gestionar esto. Es un formulario con dos botones, un botón que lanza el hilo y otro que lo detiene. También tiene una tabla para que podamos ver como se almacenan y saber si han ido a un valor o a pendientes de clasificar.

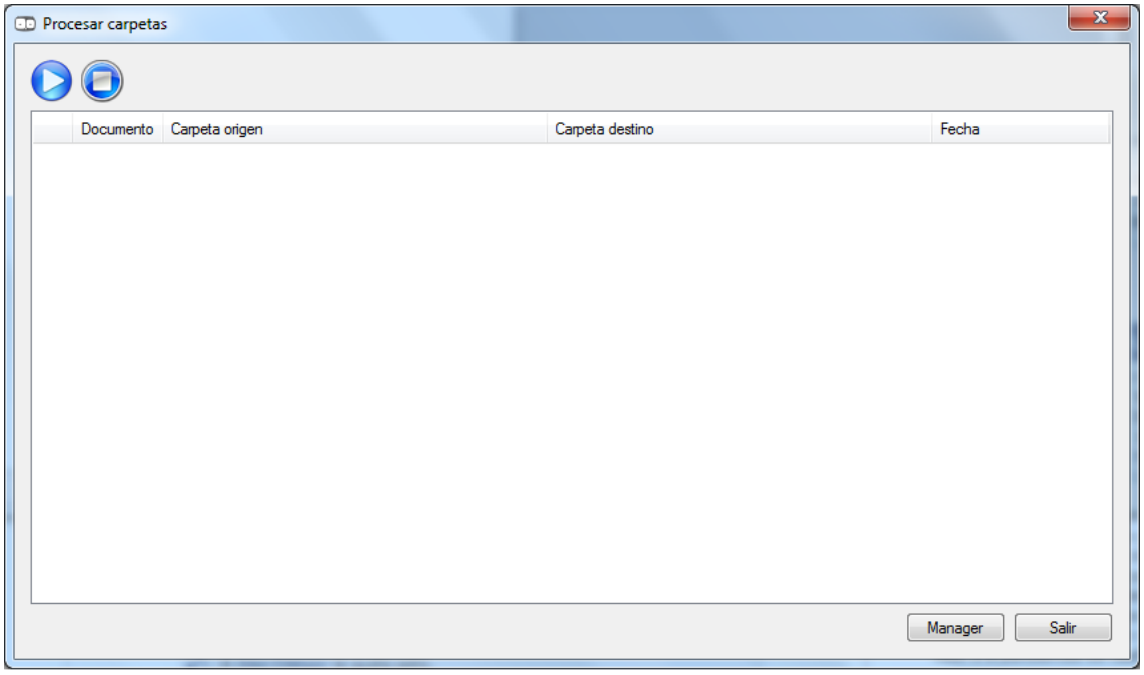

Imagen 28. Ventana que gestiona los trabajos

4.4 Pruebas.

La prueba del software representa un factor crítico para determinar la calidad del software producido además de representar una revisión final de las especificaciones, del diseño y de la codificación.

Los fundamentos de la prueba del software definen los objetivos que se consideran esenciales.

El diseño de casos de prueba se centra en un conjunto de técnicas para la creación de pruebas que satisfagan los objetivos globales de las pruebas.

Durante el diseño de la aplicación en algunos campos como, por ejemplo campos en los que se debe introducir números controlamos que no pueda introducirse otros caracteres.

La aplicación en la introducción de datos puesto que es el mismo control el que no permite el avance si no es correcto el valor, haciendo innecesaria la comprobación. No obstante se han realizado pruebas para comprobar el funcionamiento de dichas tareas. Entre las pruebas que se han realizado en la aplicación destacan las siguientes:

- Realizar repetidas inserciones en categorías, valores, carpetas de entrada y trabajos que se realizaban con éxito.
- Realizar borrados y comprobar que deja borrar cuando debe borrar y que no lo permite en situaciones que no debe hacerlo.
- Realizar modificaciones de datos ya insertados y comprobar que realiza con éxito.

4.5 Conclusión.

En este capítulo se ha expuesto todos los pasos que se han seguido (mediante un orden natural) para desarrollar la aplicación. Por tratarse del desarrollo de un sistema de información, las fases de desarrollo han coincidido plenamente con las etapas de desarrollo de software.

El proceso de diseño ha consistido en especificar los requisitos y a partir de estos se creó el modelo de la aplicación. El paso siguiente fue la codificación en el que además de implementar toda la funcionalidad de la aplicación, se desarrollaron algunos componentes, se utilizaron herramientas para extraer texto, poder

ver documentos pdf y poder ejecutar varios subprocesos al mismo tiempo. Finalmente, se hicieron las pruebas pertinentes.

El proceso que se ha seguido para desarrollar la aplicación es conocido como ingeniería del software. En el desarrollo de aplicaciones actuales no basta con programadores, se necesitan ingenieros del software que dominen cualquier etapa en el desarrollo de una aplicación (análisis, diseño e implementación), que sean buenos programadores y arquitectos de sistemas.

Un desarrollador debe tener capacidad de abstracción y síntesis, debe estar pendiente de todos los detalles, ser organizador y buen planificador, y todo esto se aprende mediante un buen método.

5 Ejemplo de la aplicación.

En este capítulo se verá una pequeña demostración de la aplicación ante un escenario de ejecución. Como hipótesis de partida se supone que la aplicación ya se encuentra instalada en el ordenador.

Los nombres y documentos utilizados en esta aplicación no son reales.

5.1 Crear una categoría.

El primer paso será insertar la categoría de las facturas a introducir en el caso de que no esté creada.

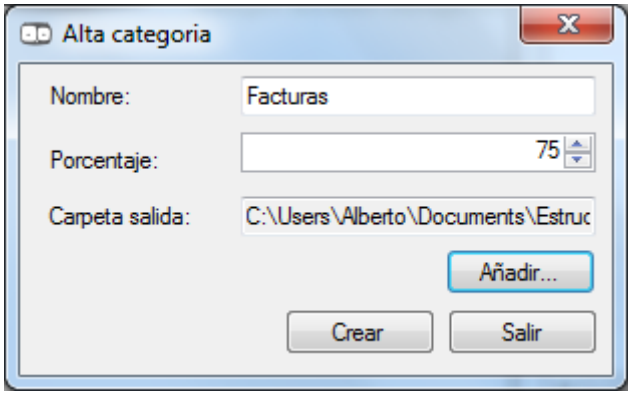

Imagen 29. Formulario para crear una categoría

Si la categoría ha sido creada con éxito se reflejará en la lista de categorías de la ventana principal.

Cuando seleccionas una categoría se rellena el listview de valores (el que está debajo) y lo rellena con los valores de esa categoría y además se activa la opción de modificar y eliminar categoría.

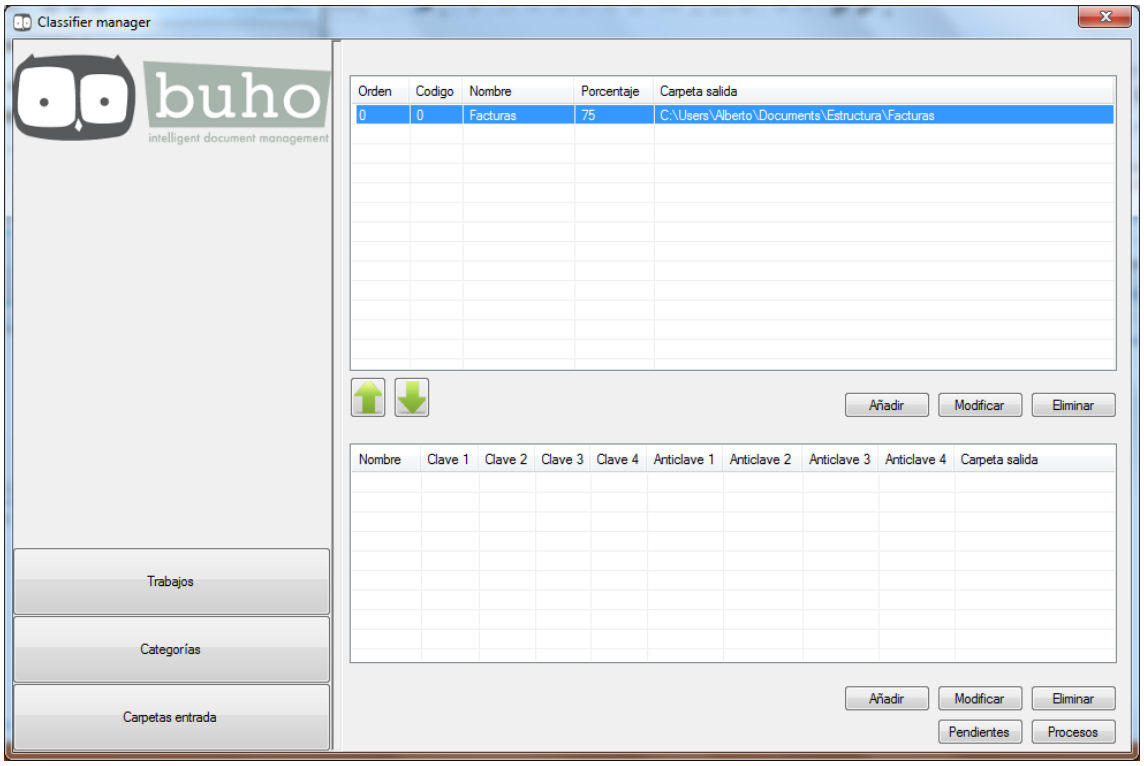

Imagen 30. Vemos la categoría creada con éxito

5.2 Crear un valor.

Para crear un valor primero debe haber creada una categoría. Seleccionamos una categoría y pulsamos el botón añadir que hay debajo del listview de valores. El botón no se podrá pulsar si no hay una categoría seleccionada.

Y rellenas los datos del formulario de valores.

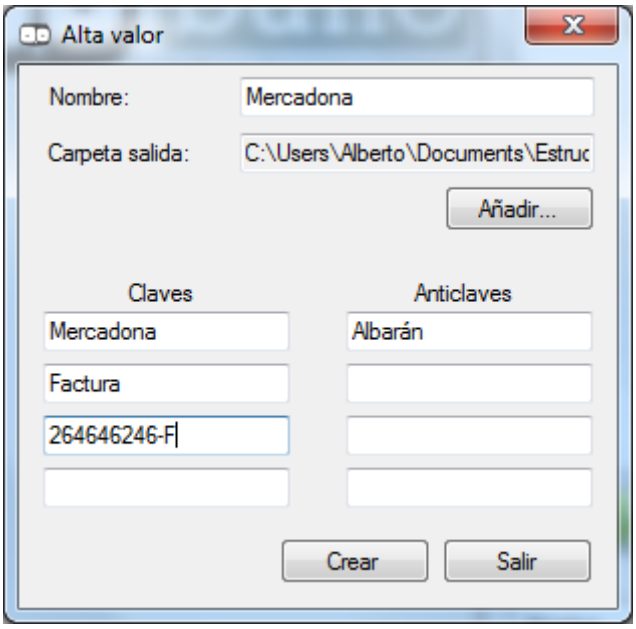

Imagen 31. Formulario para crear un valor.

En el caso de que se haya creado correctamente, aparecerá en el listview de valores que está situado debajo de categoría.

Si pulsamos sobre él se activan los botones para modificar y eliminar el valor seleccionado.

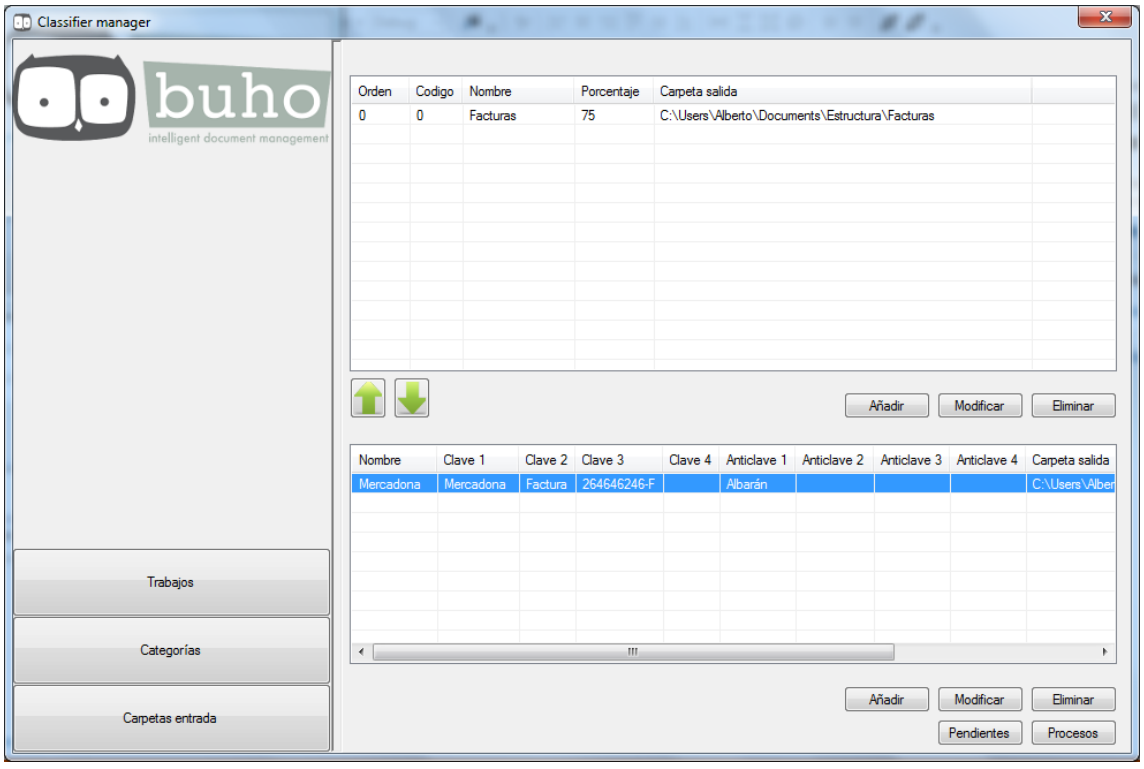

Imagen 32. Vemos el valor creado con éxito.

5.3 Crear una carpeta de entrada.

Para crear una carpeta de entrada vamos al apartado de carpetas de entrada, pulsamos el botón añadir y rellenamos los datos.

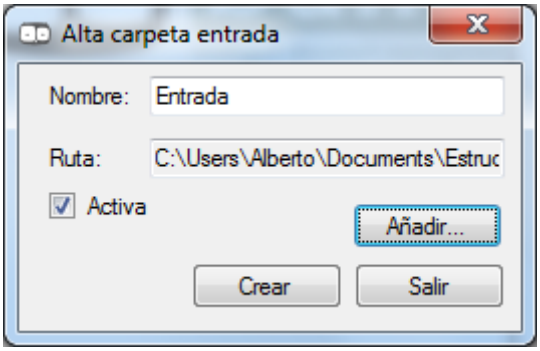

Imagen 33. Formulario para crear una carpeta de entrada.

En el caso de que se haya creado correctamente, aparecerá en el listview de carpetas de entrada.

Si pulsamos sobre la carpeta de entrada se activan los botones para modificar y eliminar la carpeta seleccionada.

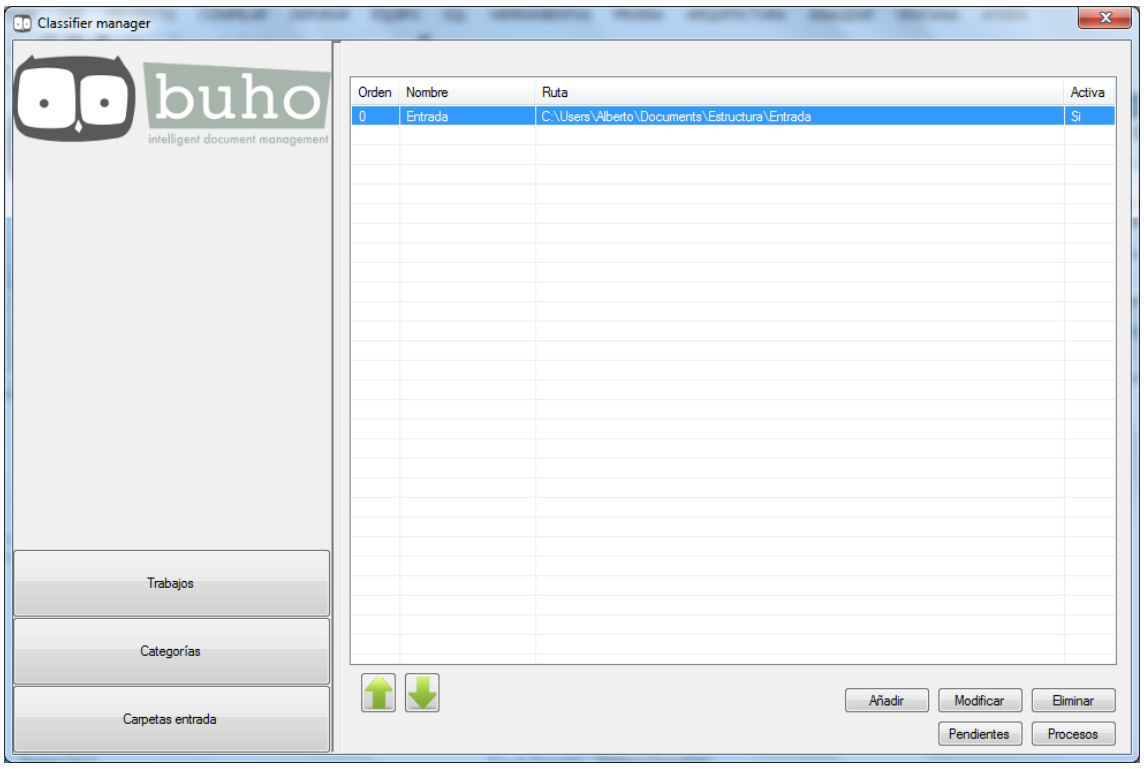

Imagen 34. Vemos la carpeta de entrada creada con éxito.

5.4 Crear un trabajo.

Para crear un trabajo vamos al apartado de trabajos, pulsamos el botón añadir y rellenamos los datos.

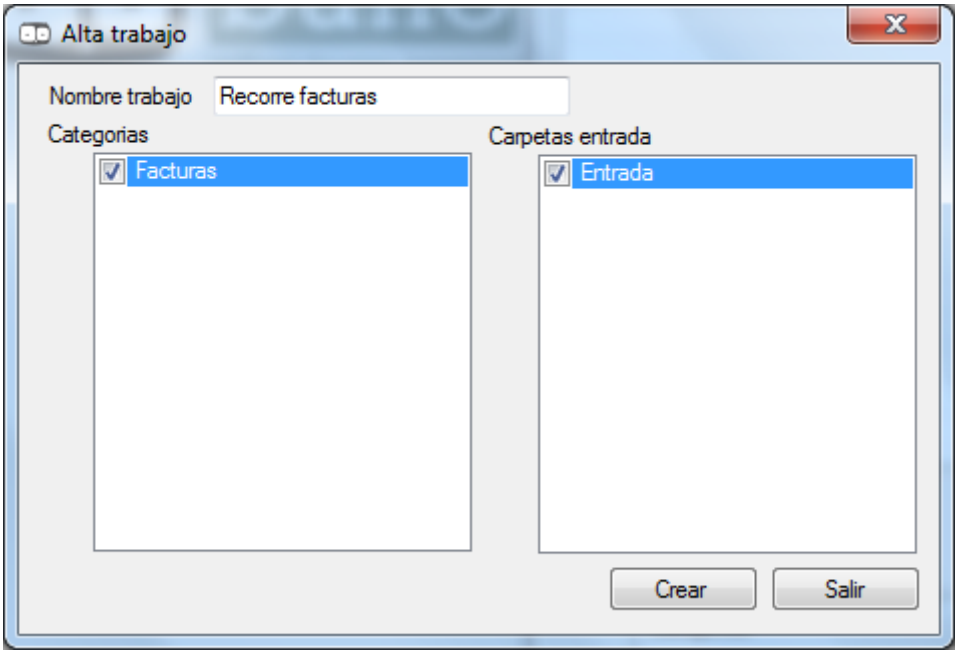

Imagen 35. Formulario para crear un trabajo.

En el caso de que se haya creado correctamente, aparecerá en el listview de trabajos.

Si pulsamos sobre el trabajo se activan los botones para modificar y eliminar el trabajo seleccionado.

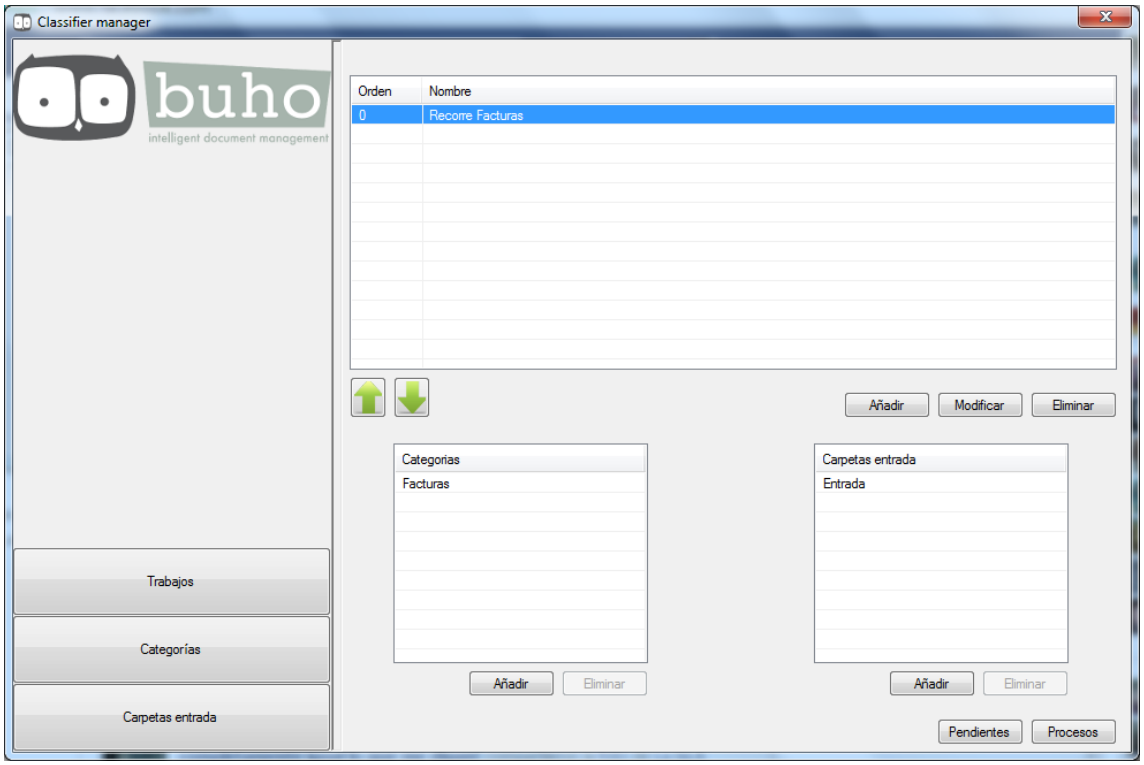

Imagen 36. Vemos el trabajo creado con éxito.

5.5 Clasificar documentos.

El siguiente paso en la aplicación sería clasificar los documentos que se quieran introducir al archivo.

Para realizar un ejemplo completo hemos creado 3 archivos pdf y un nuevo valor para que se clasifiquen los archivos donde corresponde y uno valla a pendientes de clasificar porque el programa no encuentra donde clasificarlo.

Lo siguiente será añadir los documentos en una carpeta de entrada activa y que esté incluida en un trabajo.

Y por último ir al formulario de procesar carpetas y presionar el botón “play”.

Una vez clasificados los documentos podemos ver como se han clasificado.

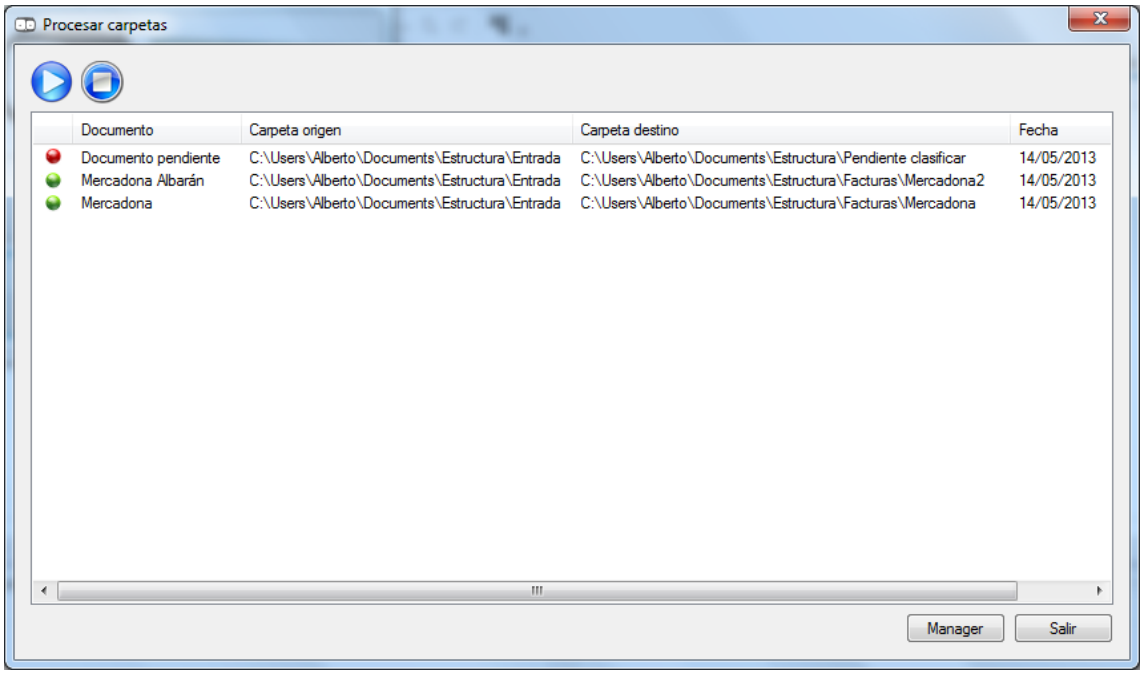

Imagen 37. Vemos como se han clasificado los documentos.

Como podemos ver en la tabla se han clasificado los documentos. Dos documentos se han clasificado satisfactoriamente, mientras que el otro no se ha clasificado.

Nuestro proyecto incluye un visor de documentos para ver los documentos que contiene cada valor.

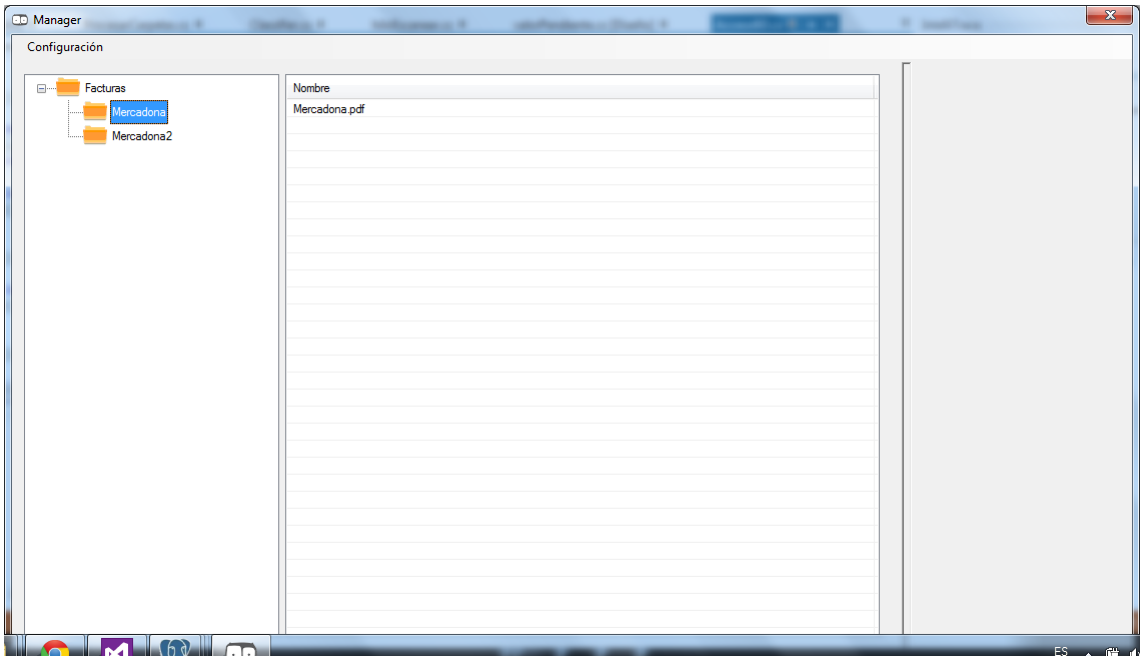

Imagen 38. Documentos incluidos en un valor.

5.6 Documentos pendientes.

Como hemos visto en el apartado anterior había un documento que no se había clasificado. Cuando no se clasifica un documento hemos de ir al formulario "pendientes clasificar" ya que hay es donde vemos los documentos no clasificados.

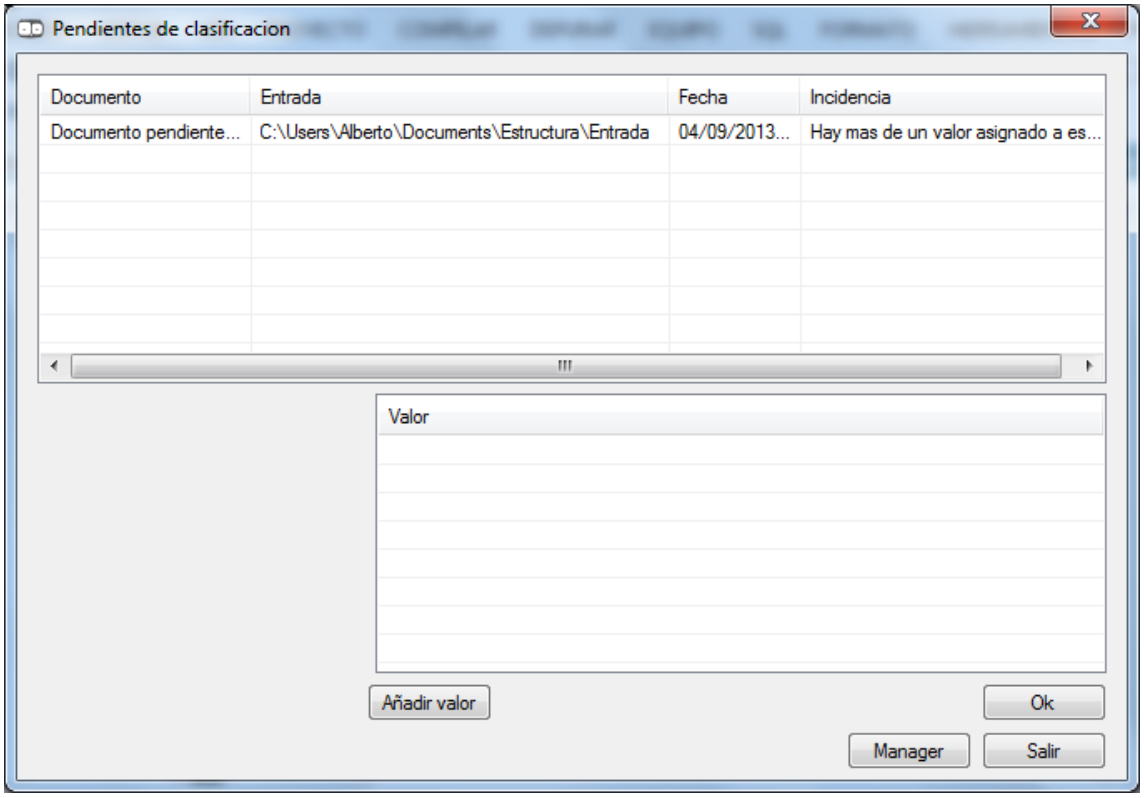

Imagen 39. Formulario pendientes de clasificar.

Como vemos en la tabla, hay un documento sin clasificar. Si observamos la tabla nos aparece la información del documento sin clasificar.

Para clasificar el documento hemos de seleccionar el documento y aparecerá un visor con el documento y los posibles valores a lo que podemos asignar el documento.

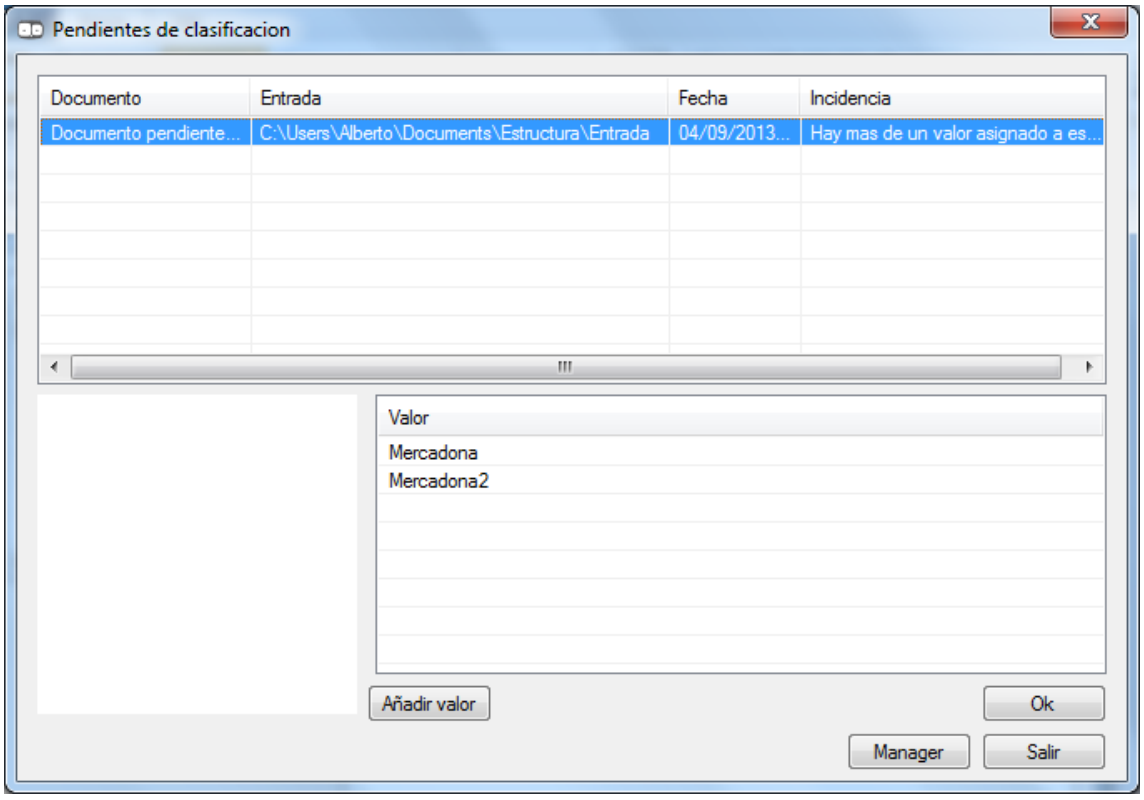

Imagen 40. Posibles valores del documento.

Para mover el documento al valor, lo seleccionamos y pulsamos el botón ok.

Si vemos la tabla de procesos podemos ver como se ha clasificado.

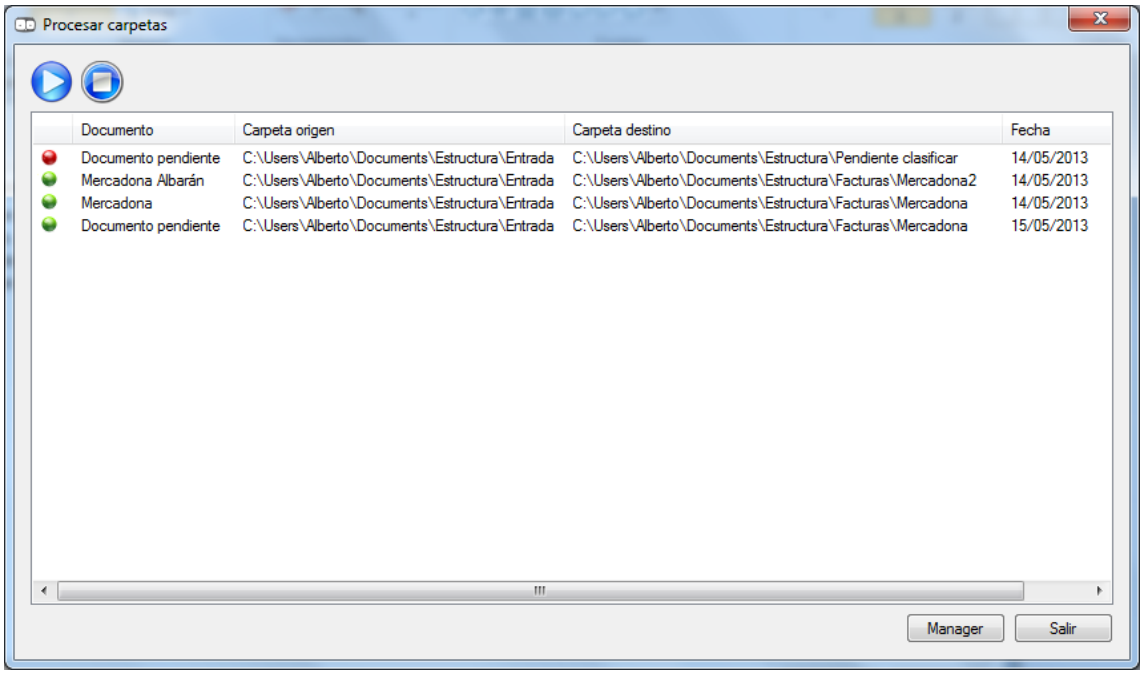

Imagen 41. Vemos la tabla de procesos.

6 Conclusiones y posibles ampliaciones.

En este capítulo se va a exponer lo que ha significado la realización de este proyecto para el alumno, así como las posibles ampliaciones del software desarrollado en este proyecto.

6.1 Beneficios del proyecto.

La creación de este sistema informático para una empresa proporcionaría unos beneficios obvios. Es evidente la mejora en cuanto a rendimiento, velocidad y facilidad de acceso que se obtendría. Más concretamente, podría obtener los siguientes beneficios:

- Más facilidad a la hora de buscar documentos.
- Menor espacio físico de almacenamiento.
- Reduciría el coste de mantenimiento.
- Más orden a la hora de almacenar los documentos.
- Ahorraría considerablemente el tiempo de clasificación de documentos.
- Ahorro en el coste de material de oficina.

6.2 Posibles ampliaciones.

Aunque este proyecto se ha basado en la gestión documental, han faltado gran cantidad de funciones ya que la gestión documental es un campo muy grande.

Algunas posibles ampliaciones serían:

- Crear un buscador que buscara por el texto extraído de los documentos.
- Una pantalla de sellado para poder sellar algunos documentos.

- Añadir un módulo de workflow para automatizar algunos procesos.
- Añadir al proyecto usuarios y empresas para hacerlo más eficiente al haber varios usuarios trabajando. También podríamos añadir permisos para cada usuario.
- Añadir un OCR zonal para poder buscar palabras concretas y buscar información concreta de algunos datos.
- Por último, añadir un módulo web o alojarlo en la nube para tener una máxima disponibilidad desde cualquier sitio.

6.3 Consideraciones finales.

Para realizar esta aplicación se han usado los conocimientos adquiridos en las asignaturas de la carrera, especialmente las que tienen que ver con la Ingeniería del Software y también los conocimientos adquiridos en las prácticas en empresa. También ha sido importante la inestimable ayuda de los profesores de la Escuela Técnica Superior de Informática Aplicada (ETSIA) a lo largo de toda la carrera. El resultado de aplicar los conocimientos de estas asignaturas y un trabajo constante han hecho posible que este proyecto haya sido viable.
Por último indicar que, a nivel personal, la realización de este proyecto me ha aportado gran cantidad de conocimientos, me ha permitido reforzar nociones de análisis y desarrollo y enfrentarme a un problema real.

7 Referencias.

[1] G. Booch. El lenguaje unificado de modelado. Guía de usuario. Addison-Wesley Iberoamericana, 2000.

[2] T. Budd. Introducción a la programación orientada a objetos. Addison-Wesley Iberoamericana, 1994.

[3] M. Celma, J. C. Casamayor y L. Mota. Bases de datos relacionales. Servicio de Publicaciones U PV. SPUPV-97.767, 1997.

[4] P. Chen. The entity-relationship model - Toward a unified view of data. ACM Transactions on Database Systems, 1976.

[5] IEEE. Guide to software requirements specifications. www.ieee.org.

[6] OMG. Object Management Group. www.omg.org.

[7] OMT. Metodologias OMT (Rumbaugh). www.monografias.com/trabajos13/metomt/metomt.shtml.

[8] B. Shneiderman. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison Wesley, tercera edición, 1998.

[9] Sommerville. Ingeniería del software. Addison-Wesley, 2002.

[10] Ayuda de Microsoft office online. http://office.microsoft.com.

[11] Ayuda de Microsoft c#. http://msdn.microsoft.com

[12] C#.5.0.Pocket.Reference (2012.5) .Joseph.Albahari.文字版.

[13] Documentación sobre PostgreSQL. http://www.postgresql.org

[14] Documentación variada. http://stackoverflow.com/

[15] PDF Parser. http://www.ajpdsoft.com.

[16] Documentación variada. http://www.codeproject.com

[17] OCR http://www.iti.es/media/about/docs/tic/02/2003-11-ocr.pdf