UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

PROYECTO FIN DE CARRERA

"TITULO DEL PROYECTO FIN DE CARRERA"

Vº Bº del Director del Proyecto	**AUTOR:** **DIRECTOR:**

NOTA MUY IMPORTANTE: Esta plantilla es sólo *una guía orientativa*. Un PFC no tiene por qué tener ni todas sus secciones ni sólo las que aparecen aquí. La plantilla debe ser adaptada a cada PFC (añadiendo secciones, quitando algunas o modificando los contenidos que aparezcan según cada caso particular), y *el alumno debe consultar siempre a su director de PFC* ante dudas acerca de cualquier tema.

Esta es la versión LaTeX[2] de la plantilla de PFC. Es casi equivalente a la plantilla Word, ya que se ha tratado de traducir todos los elementos que tiene dicha versión a este formato de la forma más fiel posible. No pretende ser un manual exhaustivo de LaTeXni mucho menos, sino un documento donde, aparte del formato y secciones que forman parte de un PFC, se recojan los elementos más comunes a usar en una documento de esta clase (listas, tablas, imagenes, etc.) y de esta forma sea mucho más facil desarrollar una documentación en LaTeX, ya no solo debido a que el documento tiene una estructura que ayuda al desarrollo de la documentación, sino también porque da solución a problemas comunes que se pueden encontrar y trata de facilitar el trabajo con LaTeXen la medida de lo posible. No obstante, algunos elementos no se han podido trasladar de forma 100 % fiel (por ejemplo, las hojas Excel incrustadas), aunque se ha hecho lo posible por proporcionar un sustituto adecuado.

NOTA: Este es un documento en constante evolución y de gran tamaño, por lo que no es raro que se puedan encontrar fallos. Si detectases algún fallo, te pedimos que colabores con los autores de esta plantilla y mejórala, enviando un correo con tus incidencias al usuario redondojose del correo de Uniovi Directo. También puedes colaborar aportando nuevas ideas, secciones, contenidos o cualquier cosa que creas que le haga falta a este documento para mejorar sus contenidos actuales. Envía un correo a la dirección anterior si crees que puedes tener algo que aportar. GRACIAS.

BORRAR ESTAS NOTAS EN LA DOCUMENTACIÓN FINAL

Agradecimientos

Esta sección no es en absoluto obligatoria, pero es el lugar correcto para dedicar el proyecto a las personas/instituciones/empresas/... que se desee.

Resumen

Un texto breve (una cara aproximadamente) que describa qué se ha hecho en el proyecto, sus principales objetivos, la utilidad que se le quiere dar, si está destinado a algún cliente real, aspectos sobre la tecnología usada y cosas similares que permitan hacerse una idea rápida del trabajo realizado.

Se trata de describir brevemente todos los aspectos más importantes del proyecto destacando en lo posible sus puntos fuertes para permitir comprenderlo fácilmente en una lectura rápida sin tener más referencias del mismo. Por tanto, no debe ser un texto demasiado largo ni complejo.

Palabras Clave

Palabra1, Palabra2, Palabra3,...

De 5 a 7 palabras[1] clave que mencionen conceptos de capital importancia en el proyecto: Cosas que el proyecto manipula (Ej.: "Máquinas Expendedoras", "Automóviles"), tecnologías usadas (Ej.: J2EE, RMI), utilidad del proyecto (Ej.: Gestión de Existencias), temática (Ej.: Historia Medieval, Aeronáutica) y cosas similares.
Si finalmente saliesen demasiados términos, conviene hacer una selección de los más relevantes para quedarse con el número indicado.

[1] No necesariamente debe ser una única palabra, pueden ser varias. Por ejemplo, si el proyecto tratase sobre la gestión de calificaciones de Alumnos, una palabra clave válida podría ser "Expediente Académico".

Abstract

Traducción al inglés del resumen anterior. Conviene hacerlo una vez se tenga la versión definitiva de dicho resumen. Se recomienda consultar al director del proyecto acerca de si considera adecuado que aparezca esta sección.

Keywords

Ídem a la sección anterior. Se recomienda consultar al director del proyecto acerca de si considera adecuado que aparezca esta sección.

Índice general

Índice de figuras

Capítulo 1

Memoria del Proyecto

El concepto de memoria de un proyecto es, en esencia, un resumen del proyecto para personas que desconozcan o no posean conocimientos avanzados de la naturaleza del proyecto y/o sus tecnologías, o incluso no posean conocimientos específicos de informática. Por tanto, debemos orientarla de manera que cualquier persona pueda entender que se ha hecho durante todo el proyecto.

Los puntos obligatorios de la memoria varían mucho de unos proyectos a otros, pero en este documento se proponen unos mínimos. En muchos casos, la memoria tiene un apartado por cada parte importante del proyecto, por ejemplo (Introducción, Requerimientos, Análisis y Diseño, Presupuesto, etc.) y en cada apartado se resume (para el perfil de lector mencionado anteriormente) el contenido del apartado técnico correspondiente. En cualquier caso, podemos orientar la memoria de la siguiente forma:

1.1. Resumen de la Motivación, Objetivos y Alcance del Proyecto

...

1.2. Resumen de Todos los Aspectos

Resumen de todos los aspectos del proyecto comentados en las secciones posteriores, tal y como se dijo anteriormente.

1.3. Otros Apartados

Otros apartados que el alumno o el director del proyecto consideren relevantes.

Capítulo 2

Introduction

2.1. Project Justification

En esta sección se debe describir el motivo por el que se desarrolla el proyecto. Se trataría de describir brevemente, en lenguaje no técnico, de que va a consistir el proyecto y el motivo de su desarrollo, incluyendo la descripción de las necesidades que va a cubrir. Se necesita hacer entonces un resumen que describa las aplicaciones del proyecto y que justifique su creación, procurando compararlo favorablemente con otras posibles soluciones.

Figura 2.1: Ejemplo de Figura

La imagen de más arriba muestra un ejemplo de cómo se pueden colocar las imágenes en todo el proyecto, incluyendo el estilo de su descripción (que está vinculado con la lista de figuras de más arriba) y la forma de numerar las imágenes propuesta (nº de capítulo – nº de imagen dentro del capítulo).

Twice a year the University of Oviedo School of Computer Science must publish the exam calendar for the around 40 subjects available at it. Each subject coordinator can state an exam per exam type, such are Theory, Practice, or other kind of examinations depending on the subject teaching guide.

It is not feasible to provide a random assignation of the exams due to the students' enrolments. The schedule must ensure that they are able to attend to their corresponding exams. Moreover, subject coordinators tend to establish restrictions to when their exams must take place. For instance, an exact date for the test could be give beforehand, a relative date comparing with other exams, or even the negation of the first case, the fact that an exam cannot take place on a specified date.

This task has been done by the deputy director for many years. Even though that the

task could seem quite straightforward, because in the end is just a matter of assigning dates to the exams, the problem is the huge number of constrictions to be considered. Furthermore, the fact that the students' enrolments change every year makes impossible to state a perfect solution to be repeated and forces constantly a new computation.

If every constriction could be understood by a machine that could propose a good enough solution the deputy director task would transform into choosing among a set of provided options and solving potential conflicts that could have arisen. In the end the task would be noticeably simplified and so the process would be much more efficient.

Since similar problems have been solved for last years, this project is aimed to find whether a solution for this actual current problem can be found.

2.2. Project Goals

The first aim of this project is to study if a solution for the University of Oviedo School of Computer Science exam calendar problem can be found. To determine such a thing, research will be done to observe how similar problems were solved in the past.

Based on how previous problems were formalized, our problem must be formalized. Thus, we will need to establish the search space, the state representation, and how we compare the possible solutions. This will require further explanation later.

Once the problem is formalized an algorithm to solve the problem will be designed. For this prototype definition process, the current files used at the University of Oviedo School of Computed Science will be considered, as well as the actual exam calendar format. The main aim of this prototype is to provide information about the quality of the assignment solutions computed, not only to check if a solution can be truly found, but also to improve the algorithm in case it is possible.

To perform the tests the algorithm will be using real data from previous exam calls, to see the behavior in real life scenarios. Those results will need to be evaluated manually in order to see the performance of the algorithm.

In the end, the prototype and the research results will be provided to the University of Oviedo School of Computer Science for its use or further extension.

2.3. Analysis of the Present Situation

En esta sección deben identificarse y describirse sistemas similares al que se va a desarrollar, estableciendo una comparación entre lo que ofrecen estos sistemas y lo que pretendemos lograr con el proyecto, para de esta forma diferenciar nuestro desarrollo de lo ya existente.

No tienen por qué ser sistemas que hagan lo mismo que el nuestro, sino que pueden ser sistemas que contengan funcionalidad en común con una parte significativa o bien que estén orientados a un conjunto de potenciales usuarios similar.

En esta sección también es adecuado evaluar las posibles herramientas o lenguajes de programación utilizables para el proyecto y determinar cuál (o cuales) se adaptan mejor a nuestras necesidades concretas (de forma justificada).

Conviene en general destacar los puntos en común y las principales discrepancias entre estos sistemas y el nuestro, con la idea de ver en qué sentido nuestro desarrollo supone una ganancia o mejora sobre ellos (también puede orientarse a resolver ciertos defectos de los mismos, mejorar algunas funciones para hacerla más completa, rápida o fácil de usar, etc.). Si los sistemas carecen de alguna funcionalidad que el nuestro va a incorporar, conviene también destacarlo (precisamente esto puede ser una de las principales aportaciones del mismo).

En general, conviene usar esta sección como un primer paso para "promocionar" las "bondades" de nuestro proyecto.

At this very moment, this problem must be solved twice a year by the deputy director. There is no automatic approach that provides a solution to the problem, not even a partial one. Therefore, the deputy director must study each case individually in order to perform manually the assignments for the exams.

It is an expensive approach, that is solved mostly based on experience and intuition. However, even an experienced user can make mistakes when the number of restrictions is as high as is in this problem.

Moreover, considering the possibility that we are forced to consider a new restriction once the planification is done, this would imply many on cascade changes, that could be as complex to handle as making a whole new assignment.

This project will be focused on providing a better approach to solve the problem in a way as automatized as possible. Due to the type of the problem, it could be the case that there is no good approach, because the constrictions are too restrictive. In such cases, the aim will be finding a good enough solution. In the end, the main target of the project is the simplification of this problem, or its complete resolution, so that the deputy director does not have to handle it in a fully manual approach.

2.3.1. Evaluación de Alternativas

En esta sección se describirán, una por una, todas las alternativas estudiadas. Conviene estudiar 3 o 4 alternativas importantes, salvo que por algún motivo justificado se deba incluir un número menor o mayor de las mismas. En todo caso, siempre es conveniente cuidar de que en esta sección haya un conjunto de sistemas significativo. En función de lo dicho anteriormente, cada sistema podrá dividirse en tres secciones: "Descripción", "Ventajas" e "Inconvenientes", aunque es posible cualquier otra división que contenga los aspectos descritos, dependiendo de qué tipo de sistemas se estudien.

A lot of literature has been published regarding scheduling problems. The main issue is that the formalization and solution of this type of problems strongly depend on the domain constraints to be considered.

Despite that fact, it seems to be some agreement on using genetic algorithms to obtain good enough solutions, as can be be seen in [paper]. Furthermore, its use has been proven effective in a previous Final Degree Project, which was also focused on the school as domain.

Nevertheless it is interesting to consider other approaches as well as the reasons for which they were not chosen.

Constraint Programming

Constraint programming is a programming paradigm in itself. It is focused on solving combination search problem such as is the machine assignment one.

Our problem differs a bit from the classical combination problem, in the end it is a scheduling problem which consists in no more than making assignments according to some criteria, such as is done in the classical problems.

Because of that it would be feasible to achieve a solution by means of Constraint Programming for the given problem, however it has some drawbacks.

- programming paradigm
- backtracking
- logic programming
- History -¿CHIP, ...

Sistema 2

...

Capítulo 3

Aspectos Teóricos

Esta sección está destinada a describir brevemente aquellos conceptos, herramientas y tecnologías existentes que vamos a usar en nuestro proyecto. Conviene limitarse a un máximo de 2 hojas por cada uno de ellos aproximadamente (es una descripción, no un tutorial sobre ello), describiendo por qué y para qué usamos esto en nuestro proyecto pero sin hacer descripciones muy grandes de características y similares (para ello podemos referenciar enlaces Web, libros, etc. que las describan más detalladamente, que posteriormente podemos incluir en la bibliografía):

1. En el caso de conceptos sobre los que trata el proyecto, debemos decir su origen, aplicación e importancia, para tener una idea de lo que se va a tratar. A modo de ejemplo, si usamos bases de datos orientadas a objetos debemos decir que son, que importancia tienen y describir brevemente su forma de trabajo. Si nuestro proyecto es sobre emisión de video en directo, debemos describir la técnica usada para ello y como funciona, etc.

2. En el caso de tecnologías debemos decir su creador u origen, para lo que sirve y que aplicaciones ha tenido, para qué la vamos a usar, la versión que hemos empleado, etc.

3. En el caso de herramientas debemos decir su fabricante y la utilidad que le vamos a dar dentro de nuestro proyecto.

4. Dentro de este apartado también podemos incluir las notaciones empleadas para representar los diagramas del proyecto (como UML) o la metodología usada para el desarrollo de la documentación (como Métrica 3). Este documento está desarrollado usando la metodología Métrica 3, pero de forma adaptada, contemplando solo aquellos apartados que se han considerado más adecuados para un proyecto de fin de carrera. No obstante, conviene hacer una reseña a esta metodología si se usa esta plantilla.

Por último, debemos recordar describir de esta forma todo aquello que usemos en el proyecto y dar especial importancia a todo aquello que sea “novedoso”.

3.1. Concepto 1

Definición

3.2. Concepto 2

Definición

Capítulo 4

Planificación del Proyecto y Resumen de Presupuestos

NOTA: Consultar al director del proyecto acerca de si considera conveniente la inclusión de una sección como esta en la documentación del PFC o que partes de la misma se deben incluir.

4.1. Planificación

En esta sección, se debe describir el plazo de ejecución del proyecto de forma que puedan fijarse las expectativas de quienes van a recibir el producto resultado del mismo.

Debe contener un cronograma explicitando las entregas parciales, hitos intermedios y duración del proyecto a partir de la fecha de iniciación del mismo. Esto debe realizarse mediante alguna metodología de gestión de proyectos, que sirva de guía sobre la descomposición en tareas, la asignación de recursos, etc.

Es buena idea incluir diagramas que describan la planificación del avance del proyecto, por ejemplo es posible utilizar un diagrama Gantt como el que puede verse en la figura 4.1:

Figura 4.1: Ejemplo de diagrama de Gantt

En función del tipo de proyecto, también es posible utilizar otros tipos de cronogramas, como el mostrado en la figura 4.2:

Figura 4.2: Ejemplo de cronograma

4.2. Resumen del Presupuesto

Explicitar el presupuesto de cliente resumido que se ha previsto para la ejecución, de forma que pueda tomarse la decisión de proseguir o no con la ejecución de lo valorado. **Esta sección puede o no tener sentido en función del tipo de proyecto desarrollado. Por ello, se aconseja consultar al director del proyecto acerca de este tema antes de hacer nada relativo a la misma.**

- Debe incluir el coste total de la ejecución para la organización que ha de hacerse cargo de este proyecto.
- En este apartado debe tenerse especial cuidado en presentar las cifras de manera no ambigua, completa, sin costes ocultos y dando un total general desglosado por partidas.
- Habitualmente sólo se ponen los ítems, eliminando del presupuesto los subitems.
- Si aún así resultase excesivamente complejo, se puede resumir en grandes partidas de trabajo que tengan sentido práctico y económico para el cliente.
- Si se usase cualquier otro criterio para hacer este resumen del presupuesto, deberá justificarse adecuadamente. La siguiente tabla muestra un ejemplo de cómo se puede realizar el mismo.
- Los formatos de la taba anterior no son obligatorios, deben adaptarse a cada proyecto.

Item	**Concepto**	**Cantidad**	**P. Unidad**	**TOTAL**
0	*Desarrollo de la Aplicación*	1,00	7500,00 €	7500,00 €
1	*Formación*	4,00	2000,00 €	8000,00 €
2	...			0,00 €
3				0,00 €
4				0,00 €
			Subtotal	15500,00 €
			IVA(16 %)	2480,00 €
			TOTAL	17980,00 €

Cuadro 4.1: Tabla de ejemplo de resumen de presupuestos

Capítulo 5

Análisis

Este apartado contendrá toda la especificación de requisitos y toda la documentación del análisis de la aplicación, a partir de la cual se elaborará posteriormente el diseño.

5.1. Definición del Sistema

5.1.1. Determinación del Alcance del Sistema

Se trata de describir de nuevo el sistema, pero en lugar de repetir lo que ya hemos dicho de él, tenemos que constatar en este apartado hasta donde vamos a llegar en su construcción, es decir, qué límites vamos a poner en el desarrollo estableciendo qué se va a hacer y qué se va a omitir (en general, hasta donde se va a llegar). Podemos por tanto usar todo lo que hemos dicho en descripciones anteriores para ayudar a describir el alcance del sistema. Conviene dejar claro este apartado para así delimitar la labor de análisis y diseño que vamos a hacer a continuación y evitar así no describir aspectos que se han construido o describir cosas que finalmente no van a construirse.

En el caso de que quede claro implícitamente qué se va a hacer en el sistema, esta sección se puede omitir.

5.2. Requisitos del Sistema

5.2.1. Obtención de los Requisitos del Sistema

El producto de esta sección se crea para su aprobación formal, es decir, que los potenciales clientes deben ver a partir de él las especificaciones completas del sistema. Además, esta sección construirá una base para solicitar cambios en los requisitos antes de avanzar más en la construcción del sistema.

Los requisitos del sistema se deben mostrar en una tabla como la que se presentará a continuación con ejemplos, ordenados por algún criterio lógico en función de a lo que se refieran. Por ejemplo, si tenemos usuarios tiene sentido agrupar todos los requisitos que tengan que ver con los usuarios.

Tampoco es necesario crear una única tabla para todos los requisitos, pueden crearse varias tablas que agrupen los requisitos que se refieran a una entidad.

Este apartado debe incluir también antes de la tabla de requisitos, si existe como tal, la **especificación textual** que el cliente nos proporcione sobre la aplicación, fruto de las reuniones que hayamos tenido con él o de las entrevistas que podamos haber llevado a cabo. Ha de tenerse en cuenta que esta información es la que usaremos para extraer los requisitos de la aplicación, por lo que no debe faltar sin contamos con ella.

Código	Nombre Requisito	Descripción del Requisito
R1.1	Insertar Usuario	Se debe añadir un usuario al sistema una vez leídos y validados sus datos.
R1.2	Leer Datos Usuario	Deben pedirse los datos completos de un usuario del sistema

Cuadro 5.1: Tabla de ejemplo de requisitos

Esta tabla puede usarse para todos los **requisitos funcionales** (lo que la aplicación debe hacer), pero debemos especificar también otro tipo de **requisitos no funcionales**, de los que a continuación se muestran ejemplos típicos:

1. **Requisitos de Usuario**: Si exigimos al usuario algún tipo de conocimiento previo para manejar la aplicación o alguna de sus partes, debemos especificarlo aquí.

2. **Requisitos Tecnológicos**: Si el programa establece que debe funcionar con una versión concreta de un determinado programa o sistema, o bien en un entorno o sistema operativo concreto debemos también hacerlo constar.

3. **Requisitos de Usabilidad**: Normas de usabilidad que la aplicación debe cumplir obligatoriamente (nosotros podemos especificar requisitos adicionales a los que el cliente solicite y desarrollarlos en la sección correspondiente).

4. **Requisitos de Seguridad**: Si debemos implementar algún tipo de medida de seguridad en el sistema (encriptación de datos, etc.).

5. **Requisitos de Tiempo de Respuesta**: Si el sistema debe proporcionar una respuesta en un tiempo acotado.

5.2.2. Identificación de Actores del Sistema

Un actor es algo o alguien que reside fuera del sistema y que interactúa con el mismo (actor primario) o bien es algo o alguien sobre el que el sistema actúa (actor secundario). Un actor puede ser una persona, un dispositivo, otro sistema o un subsistema.

Los actores representan los roles que tienen entidades ajenas al sistema pero que se relacionan con el mismo: Un mismo individuo físico puede estar representado por varios

actores en el sistema si dicho individuo tiene varios roles en el mismo. Los actores se usarán luego en la especificación de los casos de uso.

A la hora de identificar actores debe analizarse la aplicación y sus usos. Ejemplos de posibles actores son: Usuario anónimo, usuario registrado, administrador,...

5.2.3. Especificación de Casos de Uso

Un caso de uso es una descripción del comportamiento de un sistema cuando responde a una petición que se origina fuera del mismo (por parte de los actores del sistema). Un caso de uso describe "quién" puede hacer "qué" con el sistema. La creación de casos de uso se utiliza para capturar los requisitos funcionales del sistema y su comportamiento, haciéndose esto último a través de los escenarios que forman parte del mismo.

Un caso de uso se representa como una secuencia de pasos simples iniciadas por un actor de los identificados en la sección anterior, el cual interactúa con el sistema para llevar a cabo algún objetivo específico. A modo de ejemplo se pueden citar casos de uso como "*Hacer una búsqueda*", "*Enviar datos de Compras*", "*Abandonar Operación en curso*", etc. Podemos encontrar ejemplos de caso de uso adicionales y de los diagramas *UML* que representan gráficamente los casos de uso aquí:

1. `http://www.agilemodeling.com/artifacts/useCaseDiagram.htm`

2. `http://www.visualcase.com/kbase/use_case_sample.htm`

Cada caso de uso puede generar uno o varios escenarios, que describen detalladamente cada posible vía para alcanzar el objetivo del caso de uso y que serán descritos en una sección posterior. A la hora de describir casos de uso no es necesario recurrir a terminología técnica, sino que es posible usar lenguaje cercano al usuario final. Además, un caso de uso puede derivar en más subcasos de uso si es necesario describir el comportamiento del sistema con un detalle mayor.

Esta sección debe incluir el clásico diagrama de casos de uso de la aplicación. Si el número de casos de uso fuese muy elevado, se pueden crear múltiples diagramas para que quede todo de forma más clara. A modo de ejemplo, se presentan los siguientes diagramas de un sistema para la creación y corrección de exámenes de tipo test:

Figura 5.1: Ejemplo de caso de uso 1

Posteriormente, en esta sección de especificación **se deberá describir también brevemente con texto el fin u objetivo de cada caso de uso** de los diagramas anteriores usando, por ejemplo, una tabla como la siguiente. En ella se especifica el nombre único del

Figura 5.2: Ejemplo de caso de uso 2

caso de uso (se suelen nombrar según el objetivo que describen) y una descripción breve de lo que intenta hacer el usuario con este caso de uso (aunque breve, no obstante debemos asegurarnos de que es suficientemente completa y clara).

Nombre	**Descripción**
Nombre único	Descripción breve de lo que intenta hacer el usuario con este caso de uso

Cuadro 5.2: Tabla de ejemplo de casos de uso

A continuación se incluyen un par de ejemplos correspondientes a los diagramas de casos de uso de ejemplo anteriores:

Nombre	**Descripción**
Crear examen	El profesor creará un examen para que los alumnos lo hagan. El programa mostrará una pantalla que permitirá al profesor introducir las preguntas y ordenarlas, además de ver cómo va quedando el examen a medida que las introduce. Los exámenes que se pueden crear serán de uno de los tipos siguiente: ▪ Tipo test ▪ Respuestas cortas ▪ Emparejamiento
continua en la próxima página	

Nombre	Descripción

Cuadro 5.3: Ejemplo de descripción de caso de uso (I)

Nombre	Descripción
Hacer examen	El programa cargará y mostrará el examen al alumno y dará las indicaciones oportunas para su realización. Anteriormente, el alumno podrá elegir uno de los exámenes disponibles que la aplicación le ofrecerá.

Cuadro 5.4: Ejemplo de descripción de caso de uso (II)

Podemos encontrar más información en:

1. `http://en.wikipedia.org/wiki/Use_case`
2. `http://www.gatherspace.com/static/use_case_example.html`
3. `http://www.visualcase.com/tutorials/use-case-diagram.htm`
4. `http://www.wilsonmar.com/1usecase.htm`

5.3. Identificación de los Subsistemas en la Fase de Análisis

El objetivo de esta sección es analizar el sistema para poder descomponerlo en sistemas más pequeños (subsistemas) que faciliten su posterior análisis.

5.3.1. Descripción de los Subsistemas

En esta sección debemos enumerar todos los subsistemas que identifiquemos inicialmente en la aplicación. Los subsistemas son agrupaciones de paquetes y clases que tienen un objetivo propósito común. Ejemplos de subsistemas pueden ser todas las clases que manejen la base de datos (subsistema "base de datos"), clases que agrupen un conjunto de servicios relacionados, clases del cliente de esos servicios, etc.

5.3.2. Descripción de los Interfaces entre Subsistemas

Una vez identificados los subsistemas, debemos también describir cómo será la comunicación entre los mismos. Ejemplos de ello son por ejemplo el uso de un protocolo (IP, SOAP) o una API o Interfaz de operaciones. En general conviene destacar si estos subsistemas se comunicarán localmente (dentro de la propia máquina) o por red.

5.4. Diagrama de Clases Preliminar del Análisis

En la fase de análisis podemos identificar ya posibles clases del sistema, a partir de los casos de uso y subsistemas ya vistos. Estas clases no tienen por qué ser definitivas ni contener todas las operaciones y atributos que finalmente tendrán (sólo los que sean obvios según los requisitos y casos de uso), pero sirven como punto de partida para el esquema completo que se desarrollará en la fase de diseño. Por ello, estas clases no deben tener nombres de operaciones con sus parámetros exactos, sino más bien una indicación de lo que deben hacer esas operaciones. Por ejemplo, en lugar de "insertarUsuario(Usuario:u)" es mejor poner simplemente "insertar usuario". La descripción de las clases se hará agrupándolas por el subsistema al que pertenecen.

5.4.1. Diagrama de Clases

Previamente a describir las clases una por una, debemos incluir **un diagrama de clases global que muestre la relación entre todas ellas**. Se recuerda que esta es sólo la fase de análisis y no es necesario lograr un nivel de profundidad de detalle muy elevado, sino que lo que se busca es una idea aproximada (pero precisa) de cómo va a ser el sistema a construir. En este diagrama también pueden aparecer los subsistemas identificados anteriormente. A continuación se muestra un ejemplo:

Figura 5.3: Diagrama de clases de ejemplo

5.4.2. Descripción de las Clases

Las clases deberían estar organizadas por los subsistemas identificados en anteriormente, rellenando una tabla como la siguiente por cada clase del mismo:

Subsistema 1

Nombre de la clase
Descripción
Descripción del propósito de la clase
Responsabilidades
Responsabilidades de la clase
Atributos Propuestos
continua en la próxima página

Nombre de la clase
▪ **NombreAtributo**: Descripción de su propósito ▪ **NombreAtributo2**: Descripción de su propósito
MÉTODOS PROPUESTOS
▪ **NombreMétodo**: Descripción de lo que hace ▪ **NombreMetodo2**: Descripción de lo que hace

Subsistema 2

Nombre de la clase
DESCRIPCIÓN
Descripción del propósito de la clase
RESPONSABILIDADES
Responsabilidades de la clase
ATRIBUTOS PROPUESTOS
▪ **NombreAtributo**: Descripción de su propósito ▪ **NombreAtributo2**: Descripción de su propósito
MÉTODOS PROPUESTOS
▪ **NombreMétodo**: Descripción de lo que hace ▪ **NombreMetodo2**: Descripción de lo que hace

5.5. Análisis de Casos de Uso y Escenarios

En esta sección se describirán los casos de uso identificados anteriormente de forma detallada, a través de sus escenarios. Los escenarios describen las interacciones entre los usuarios y el sistema e incluyen información acerca de los objetivos, expectativas, motivaciones, acciones y reacciones que se llevan a cabo. Los escenarios deben reflejar la forma en la que un sistema se comporta y se suelen describir en lenguaje coloquial, intentando no recurrir excesivamente a tecnicismos para poder ser entendidos por el usuario final. La intención de los mismos es definir el comportamiento deseado del *software* de manera que complementen a los requisitos funcionales antes descritos. Si se está usando una metodología de desarrollo ágil, los escenarios se detallan como acciones breves del usuario.

Es también frecuente incluir escenarios que describan acciones erróneas o equivocadas que el programa debe tener en cuenta, para asegurar un comportamiento adecuado y seguro en todos los posibles casos. A la hora de incluir estos escenarios en la descripción de un sistema se deben tener en cuenta aquellos casos en los que el sistema pueda tener un posible problema de seguridad o de fiabilidad. Por ejemplo, casos en los que el usuario introduzca información errónea, se produzca algún error al hacer algún cálculo o bien temas relacionados con algunos requisitos no funcionales.

Como ya se ha dicho, los escenarios se generan a partir de los casos de uso identificados. Como mínimo debe existir un escenario primario o principal que describa el flujo normal de los eventos que transcurran en el caso de uso, es decir, lo que debe ocurrir normalmente cuando este se ejecuta. Por ejemplo, para un caso de uso "*Registro en el sistema*", la secuencia de pasos asociada al escenario principal del mismo "*El usuario se da de alta correctamente*" sería:

1. El sistema muestra la pantalla de logon.
2. El usuario introduce su nombre y clave de usuario
3. El sistema valida la información introducida
4. El usuario entra correctamente en el sistema.

Además, también existirán escenarios que describan caminos secundarios o alternativos que son variantes del principal mostrado anteriormente. Por ejemplo, para el caso de uso anterior unos posibles escenarios secundarios serían:

1. **Escenario Alternativo 1**: Alta errónea porque el usuario ya existe
2. **Escenario Alternativo 2**: Alta errónea porque la contraseña no cumple las especificaciones requeridas.
3. **Escenario Alternativo 3**: Alta errónea porque faltan campos obligatorios en el formulario.
4. etc.

Los casos de uso es frecuente que se vuelvan complejos y generen un gran nº de escenarios secundarios (pueden tener un gran nº de pasos y cada uno de ellos genera varios escenarios secundarios). Por ello, se usan diagramas de actividad o robustez para poder representar mejor esa complejidad.

Por otro lado, los escenarios alternativos no deben contemplar los errores a nivel de sistema (fichero no encontrado, error de conexión), sino que suelen incluirse en su propia sección de "Excepciones" para tratar de no repetir la misma información de errores entre escenarios. Los escenarios alternativos son más secuencias de pasos alternativas a la "normal" o "principal", pero que luego pueden estar relacionados con la misma. Por ejemplo, el escenario alternativo anterior "*Alta errónea porque faltan campos obligatorios*

en el formulario" finalizará en "*Ir al paso 1 del escenario principal*" (volver a pedir los datos de login).

Bajo esta perspectiva, el número de escenarios que caben dentro de una aplicación medianamente compleja puede ser muy elevado. Entonces, ¿dónde poner el límite? ¿Cuáles deben ser representados? El analista debe ser capaz de seleccionar aquellos escenarios que aporten información útil al diseño. Por ejemplo, en toda aplicación web cabe considerar el escenario de "fallo de conexión". Sin embargo, si no se requiere un tratamiento específico para esta circunstancia, esa información es redundante dado que más que informar, despista al diseñador. Sin embargo, si queremos que, por ejemplo, para dar de alta un cargo en una actividad de la empresa ésta deba estar abierta, y en consecuencia que esto sea controlado por el sistema en desarrollo, sí deberemos considerar el escenario "Intento de asignación de cargo a actividad cerrada" dentro del caso de uso "Asignar cargo a actividad".

Una vez vista una pequeña descripción de los conceptos involucrados en esta sección, pasaremos a describir qué elementos deberían aparecer en la documentación final para contemplar todo lo dicho. Para ello se debe describir la secuencia de pasos de la que constan los escenarios y sus alternativas, clasificándolos según el caso de uso al que correspondan de una forma similar a la que se muestra en el ejemplo siguiente. Debe además tenerse en cuenta que no hay una plantilla estándar para describir los casos de uso y sus escenarios en una documentación, sino que es frecuente adaptar su descripción al proyecto que se está describiendo. A continuación se da un ejemplo de tabla para la descripción de los mismos, que puede adaptarse reduciéndose o ampliándose en función de las necesidades:

Nombre del Caso de Uso	
Precondiciones	Descripción de todas las condiciones que deben cumplirse para iniciar el caso de uso. Esto quiere decir que si el sistema no está en estado descrito por sus precondiciones, el comportamiento del caso de uso no está determinado.
Poscondiciones	Describe que cambios en el estado del software se producirán tras completar el caso de uso
Actores	Qué actores están involucrados en el caso de uso (quién lo inicia, quién lo termina)
Descripción	Se usará para capturar la esencia del caso de uso (su escenario principal), describiendo el contenido del mismo y sus operaciones
Esc. secundarios	Aquí deben describirse todas las posibles variaciones contempladas sobre el escenario principal, es decir, la descripción de todos los escenarios secundarios identificados
Excepciones	Condiciones excepcionales o errores que puedan ocurrir en el escenario principal y/o los secundarios descritos antes
Notas	Cualquier aclaración necesaria que no se haya contemplado en los puntos anteriores

Para la documentación, los casos de uso complejos o de importancia elevada es mejor documentarlos con un diagrama de actividad o robustez además del texto que hagamos siguiendo la tabla anterior. De esta forma, se debería dividir la sección por cada caso de uso, y dentro de cada una de **ellas incluir primero un diagrama con la secuencia de pasos que contempla el mismo seguido de tablas como la mostrada antes, una por cada caso de uso**. A continuación se muestran unos ejemplos de esto para aclarar el contenido de esta sección.

5.5.1. Caso de Uso 1

NOTA: No debemos olvidarnos de incluir una explicación del diagrama con aquello que pudiese no quedar del todo claro en el mismo.

(El siguiente ejemplo de diagrama de robustez (Figura 5.4) corresponde al escenario principal del caso de uso “Crear examen” dado en un ejemplo anterior)

Figura 5.4: Descripción de las actividades de un escenario con un diagrama de robustez (I)

Crear Examen	
Precondiciones	El usuario debe estar validado con rol de profesor.
Poscondiciones	Debe existir un nuevo examen con identificador único en el sistema
Actores	Iniciado y terminado por el profesor
Descripción	El profesor: 1. Accederá a la pantalla de nuevo examen 2. Rellenará la información necesaria para confeccionar el examen 3. Asignará las preguntas al examen escogiendo entre las existentes 4. Guardará el examen 5. Recibirá una notificación del éxito en la operación

continua en la próxima página

Crear Examen	
Esc. secundarios	▪ **Escenario Alternativo 1:** El identificador del examen indicado ya existe en el sistema • Volver al paso 2 del escenario principal, manteniendo el resto de información en la pantalla ▪ **Escenario Alternativo 2:** El usuario no encuentra la pregunta que busca entre las existentes • Dar la posibilidad al usuario de editar nuevas preguntas, conectando con el escenario principal del caso de uso "Editar Nuevas preguntas" • Actualizar la lista de preguntas disponible • Volver al paso 3 del escenario principal
Excepciones	▪ La base de datos no está disponible: No se pueden obtener preguntas ni guardar exámenes • Notificar un error asociado al problema encontrado
Notas	-

5.5.2. Caso de Uso 2

(El siguiente ejemplo de diagrama (Figura 5.5) de robustez corresponde al escenario principal del caso de uso "Registro en el sistema" mencionado en un ejemplo anterior)

Figura 5.5: Descripción de las actividades de un escenario con un diagrama de robustez (II)

Registro en el Sistema	
Precondiciones	No
Poscondiciones	El usuario debe estar validado y con un rol asignado
Actores	Iniciado por un usuario de cualquier tipo de la aplicación finalizado por un usuario con el rol asociado a la información de logon introducida
continua en la próxima página	

Registro en el Sistema	
Descripción	1. El sistema muestra la pantalla de logon. 2. El usuario introduce su nombre y clave de usuario 3. El sistema valida la información introducida 4. El usuario entra correctamente en el sistema
Esc. secundarios	▪ **Escenario Alternativo 1**: Alta errónea porque faltan campos obligatorios en el formulario • Notificar el hecho al usuario • Volver al paso 1 del escenario principal ▪ **Escenario Alternativo 2**: Usuario y/o contraseña inválidos • Notificar el hecho al usuario, sin dar detalles de lo que falta por seguridad • Volver al paso 1 del escenario principal ▪ **Escenario Alternativo 3**: Usuario y/o contraseña inválidos introducidos más de 5 veces seguidas • Notificar el hecho al usuario, sin dar detalles de lo que falta por seguridad • Volver al paso 1 del escenario principal, pero cambiando el contenido de la pantalla de login por un mensaje de forma que no se pueda volver a introducir información de login • Mantener esa invalidación para la IP del usuario que se conecta durante 30 minutos

continua en la próxima página

Registro en el Sistema	
Excepciones	▪ La base de datos no está disponible: No se pueden obtener nombres ni contraseñas de usuario • Notificar un error asociado al problema encontrado
Notas	En caso de que la BBDD no esté disponible, se debe mostrar un error pidiéndolo al usuario que lo intente en unos minutos en la pantalla de login

Podemos encontrar más información en: `http://en.wikipedia.org/wiki/Scenario_(computing)`.

Por último, dado que los diagramas de robustez que se pueden usar para la descripción de la secuencia de pasos a dar en un escenario son un concepto relativamente nuevo y poco conocido, se ha incluido el siguiente anexo para su mejor comprensión.

Anexo: Diagramas de Robustez

El propósito principal de estos diagramas es poder analizar los pasos de cada caso de uso para validar que la lógica de negocio que modela es correcta, y que la terminología usada es consistente con la de otros casos de uso que se han analizado previamente. Por tanto, pueden usarse para comprobar si nuestros casos de uso son lo suficientemente robustos para representar a los requisitos del sistema construido. Otra ventaja de hacerlos es que permite identificar objetos o responsabilidades de objetos que son necesarias para soportar la lógica modelada por cada caso de uso, pero que se llaman fuera del mismo, sirviendo como puente hacia otros diagramas como diagramas de secuencia o diagramas de clase.

En un diagrama de robustez aparecen los siguientes conceptos:

1. **Actores**: Los identificados en la sesión correspondiente.
2. **Elementos limítrofes o *"Boundary Elements"*:** Representan elementos software como pantallas, informes, páginas HTML o interfaces del sistema con los que cada actor interactúa. También se les denomina "Elementos de Interface"
3. **Elementos de Control o *"Control Elements"*:** Sirven como unión entre elementos *"Boundary"* y las entidades que veremos a continuación, implementando la lógica necesaria para gestionar los elementos y sus interacciones. También se les suele denominar elementos de proceso o controladores. Es importante entender que es posible implementar controladores dentro del sistema con elementos que no sean objetos (si son muy simples, pueden representarse con métodos de una entidad o clase *"Boundary"* simplemente).
4. **Entidades o *"Entity Elements"*:** Estos son los tipos de entidad que se encuentran en el modelo conceptual ("Estudiante", "Aula", etc.).
5. **Otros casos de uso (opcional):** Dado que en muchas ocasiones unos casos de uso invocan a otros, puede ser necesario representar esto en los diagramas de robustez.

El uso de estos diagramas será el correcto si vemos que nos reportan las siguientes ventajas:

1. **Control de corrección:** Ayudan a estar seguro de que la descripción de cada caso de uso y sus escenarios es correcta y que no describe comportamientos no razonables o imposibles. En ocasiones puede ser necesario cambiar los nombres usados en la descripción del caso de uso con los nombres que aparecen en los diagramas de robustez.

2. **Control de completitud:** Ayuda a asegurarnos que los casos de uso encajan con las operaciones que pretendemos hacer.

3. **Identificación de objetos:** Es posible que nos hayamos olvidado de identificar algunos objetos en las partes del análisis hechas anteriormente y estos diagramas nos ayudarán a saberlo. También pueden ayudarnos a identificar discrepancias y conflictos entre nombres que hayamos asignado a las entidades anteriormente. En caso de encontrar estos fallos, debemos modificar los diagramas de manera acorde.

4. **Ayuda a una fase preliminar del diseño:** Estos diagramas suelen ser más sencillos y fáciles de leer que los de secuencia.

Para una realización correcta de estos diagramas debemos tener en cuenta:

1. Que las entidades que representemos en este diagrama DEBEN aparecer en el diagrama de clases de entidades del análisis hecho anteriormente.

2. Que los diagramas diagramas tienen que describir los procesos de los casos de Uso/escenarios que hayamos identificado. En caso de encontrar discrepancias, se debe identificar cual es el error y arreglarlo en aquel diagrama que corresponda.

Podemos encontrar más información en:

1. `www.agilemodeling.com/artifacts/robustnessDiagram.htm`

2. `pst.web.cern.ch/PST/HandBookWorkBook/Handbook/SoftwareEngineering/UCDOM_robustness.html` (Especialmente recomendado si este tipo de diagramas no se ha visto anteriormente, detallando además posibles errores a la hora de construir estos diagramas)

Por último, debemos recordar que si los escenarios representan operaciones muy sencillas o triviales no es necesario hacer un diagrama para los mismos. Tampoco tiene mucho sentido desarrollar muchos diagramas casi iguales; si varias operaciones funcionan prácticamente de la misma forma, bastaría con indicar que el diagrama correspondiente a la operación X es muy similar al mostrado para la operación Y, indicando en texto las diferencias. Es también importante tener herramientas que nos permitan realizar fácilmente estos diagramas. Actualmente herramientas como *DIA* (instalable mediante el "centro de *software"* de versiones recientes de Ubuntu) o *Enterprise Architect* son ejemplos de herramientas que lo permiten.

5.6. Análisis de Interfaces de Usuario

A la hora de diseñar un interfaz de usuario, debemos cumplir con las normas de comunicación persona-máquina existentes, procurando que el interfaz sea usable, permita manejar el programa de manera eficiente y que no sea propenso a provocar errores en los usuarios. Esto debe hacerse así porque los diseños obedecen al resultado de hacer un diseñado centrado en el usuario, que simplemente nos lleva a simular en la pantalla el trabajo que realiza sobre una mesa o, en general, su entorno de trabajo existente hasta el momento. Un enlace que puede ser de ayuda a la hora de tomar determinadas decisiones a la hora de construir el interface de la aplicación es el siguiente: `http://www.ambysoft.com/essays/userInterfaceDesign.html`

5.6.1. Descripción de la Interfaz

En esta sección debemos crear la especificación de las interfaces entre el usuario y el sistema a construir, incluyendo todos los diferentes tipos de pantallas que van a existir, los cuadros de diálogo o los informes que le proporcionarán al usuario.

En este apartado también es importante identificar posibles grupos de usuarios para así aplicar las pantallas a dichos grupos, así como detallar otros aspectos, como lo que vamos a incluir en las pantallas para cumplir con normas de accesibilidad y usabilidad.

Para los distintos tipos de pantallas, se debe hacer un esquema que muestre la disposición de las mismas, que permita identificar donde irá cada elemento y las diferentes zonas de trabajo. Se muestra un ejemplo con este dibujo:

Figura 5.6: Boceto de una interfaz

Otra posible opción para este apartado es diseñar ya las pantallas definitivas sin funcionalidad, solo para ver como quedarán en el producto final (es decir, crear un **prototipo**), lo que tiene la ventaja de poder enseñarle al cliente el aspecto de la aplicación desde un primer momento.

5.6.2. Descripción del Comportamiento de la Interfaz

En este apartado debemos especificar cosas como los convenios que vamos a crear para validar la entrada de datos de la aplicación, los mensajes de error que mostraremos y el tipo de ayuda que vamos a proporcionar al usuario.

5.6.3. Diagrama de Navegabilidad

En esta sección incluiremos un diagrama que muestre la navegación que habrá entre las pantallas del programa y su relación con las computaciones que tienen lugar en las mismas. Debemos mostrar solo las transiciones entre pantallas y no el contenido de cada pantalla en sí, ya que este diagrama tiene simbología especial para todos sus elementos. Podemos encontrar más información en:

- `www.agilemodeling.com/artifacts/uiFlowDiagram.htm`.

5.7. Especificación del Plan de Pruebas

En esta sección crearemos y diseñaremos el plan de pruebas de la aplicación y sus funciones, así como todos los mecanismos que utilizaremos para detectar errores y corregirlos ya en la fase de implementación.

Las pruebas contemplarán aspectos tanto de funcionalidad de la aplicación como de aspectos de los usuarios o clientes de la misma.

Se contemplarán hasta cinco tipos de pruebas:

1. **Pruebas Unitarias**: Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código, o en este caso una clase individual que cumple con una función concreta. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. A partir de los casos de uso, los escenarios y clases vistos anteriormente, debemos desarrollar pruebas unitarias que consideremos necesarias y especificar los resultados que se espera encontrar una vez ejecutada la operación sobre cada una de ellas. Es conveniente tabular estas pruebas para su aplicación posterior.

2. **Pruebas de Integración**: Las pruebas de integración comprenden verificaciones asociadas a grupos de componentes, verificando que éstos funcionan correctamente cuando estos son ensamblados para cumplir con una función concreta. Para ello, cada escenario debe probarse con el mayor número de entradas posibles (y relevantes) que sea posible, incluyéndose entradas con datos correctos y con datos incorrectos para probar que el sistema reacciona correctamente ante errores de los usuarios. Para elaborar estas pruebas debemos tener en cuenta las características de la aplicación.

3. **Pruebas del sistema**: Las pruebas del sistema son pruebas de integración del sistema construido completo, que permiten probar el conjunto de todo el sistema y que sus relaciones con otros sistemas que necesite son correctas, verificando así que todas sus especificaciones funcionales y técnicas se cumplen.

4. **Pruebas de Usabilidad**: Este tipo de pruebas determinan la satisfacción del cliente con el producto final. Podemos especificar aquí cuales de estos aspectos son los más importantes en la aplicación a crear y establecer unas pautas generales por las cuales queremos medir en qué medida hemos conseguido estos aspectos. No se trata de diseñar ya los mecanismos de prueba de estos aspectos, ya que eso se hará posteriormente.

5. **Pruebas de Código**: Para determinar la existencia de código muerto. Se recomienda hablar si incluir este tipo de pruebas o no con el director del proyecto.

Para elaborar las pruebas debemos desarrollarlas a partir de los casos de uso y escenarios antes descritos, empleando tablas como las que se muestran a continuación. Para elaborarlas debemos tener en cuenta el escenario principal del caso de uso y sus posibles alternativas y excepciones. Estas tablas sirven como ejemplo para pruebas de integración o del sistema. Si resultase más sencillo, puede hacerse con pequeñas tablas independientes para cada caso. A continuación se muestra un pequeño ejemplo para un caso de uso:

Caso de Uso 1: Añadir Usuario	
Prueba	**Resultado Esperado**
Añadir un usuario no existente	El sistema posee un usuario más
Prueba	**Resultado Esperado**
Añadir un usuario que ya existe	El sistema no posee un usuario más y se muestra un dialogo notificándolo

continua en la próxima página

Caso de Uso 1: Añadir Usuario	
Prueba	**Resultado Esperado**
Cancelar la Operación	El sistema permanece sin cambios.

Capítulo 6

Diseño del Sistema

6.1. Arquitectura del Sistema

6.1.1. Diagramas de Paquetes

Aunque el concepto de paquete se puede aplicar a varios elementos del modelo, lo más típico en un PFC es agrupar clases con ellos. Los paquetes agruparán clases que estén relacionadas con una determinada funcionalidad y este diagrama debe mostrar también las relaciones existentes entre dichos paquetes (por ejemplo, dos paquetes estarán relacionados si algunas de sus clases se usan entre ellas o se envían mensajes). En esencia se trata de mostrar la organización lógica de la aplicación (en contraposición con la organización física, que aparecerá en los diagramas posteriores), presentando como se agrupan las clases de dicha aplicación en paquetes y la relación existente entre los mismos. Más información sobre estos diagramas se puede encontrar en:

- `www.agilemodeling.com/artifacts/packageDiagram.htm`
- `www.sparxsystems.com.au/resources/uml2_tutorial/uml2_packagediagram.html`

Los diagramas de paquetes son especialmente útiles cuando el diagrama de clases de todo el sistema es demasiado grande para visualizarse correctamente. Podemos entonces hacer un diagrama de paquetes (cada paquete contendrá N clases del sistema) y luego detallar individualmente cada uno de ellos donde corresponda.

Paquete 1

Descripción de los paquetes del diagrama (que tipo de elementos contiene, para qué sirven los mismos,...).

Paquete 2

Descripción de los paquetes del diagrama (que tipo de elementos contiene, para qué sirven los mismos,...).

6.1.2. Diagramas de Componentes

Los diagramas de componentes muestran los diferentes componentes de un sistema y sus dependencias. Un componente representa un módulo de código físico. Muchas veces se suele identificar un componente con un paquete, pero no siempre es así ya que los componentes representan el empaquetado físico del código. Por tanto, una misma clase puede estar presente en varios componentes, pero definida en un único paquete.

Es también una práctica común el incluir estos diagramas dentro de los de despliegue (que veremos a continuación) para que la distribución física de las distintas partes de la

aplicación (y los componentes que las forman) en las distintas máquinas disponibles quede más clara.

Para tener más información sobre este tema, se pueden consultar los siguientes enlaces:

- `http://www.agilemodeling.com/artifacts/componentDiagram.htm`
- `http://www.visual-paradigm.com/VPGallery/diagrams/Component.html`
- `http://www.ibm.com/developerworks/rational/library/dec04/bell/`

Si se integra con el diagrama siguiente (el de despliegue), en las descripciones que hagamos de los elementos debemos describir también los componentes. En caso contrario, describiremos los componentes en esta sección.

6.1.3. Diagramas de Despliegue

El sistema creado puede estar compuesto por varios procesos *software* y máquinas que colaboran para llevar a cabo la tarea encomendada, y esta sección es la indicada para representar estos procesos y máquinas así como la relación existente entre ellos. Debemos ofrecer un diagrama (un ejemplo extremadamente simple de cómo podemos hacerlo se muestra a continuación) y posteriormente una descripción de que es cada parte de la aplicación y su función (relacionándolo con las clases y paquetes vistos en el análisis).

Figura 6.1: Ejemplo simple de arquitectura del sistema

Aunque el nivel de detalle puede variar en función del sistema diseñado, los elementos mostrados deben dejar clara la distribución de la aplicación. Podemos encontrar más información al respecto en los siguientes enlaces:

- `http://www.agilemodeling.com/artifacts/deploymentDiagram.htm`
- `http://www.visual-paradigm.com/VPGallery/diagrams/Deployment.html`

Elemento 1

Descripción de los elementos del diagrama anterior.

6.2. Diseño de Clases

En esta sección representaremos diagramas que muestren los paquetes (y la relación existente entre ellos) y las clases que formarán parte de la implementación final del sistema (incluyendo también las relaciones existentes entre ellas), explicando todo aquel dato acerca de la utilidad de los mismos y justificando todo aquello que consideremos necesario.

1. Debemos extraer la información de métodos y atributos a incluir en cada clase (así como las relaciones existentes entre ellas) de los diagramas que hemos desarrollado en el punto anterior.

2. También debemos mostrar las clases adicionales que pueden ser necesarias en función de lo que hemos desarrollado anteriormente.

3. Debemos mostrar todas las asociaciones y agregaciones (en general, relaciones) que necesitemos, en función de dichos diagramas.

4. El conjunto de atributos de las clases debemos crearlo según los métodos, relaciones y en general de las necesidades de cada clase del sistema.

5. La jerarquía de clases debemos pensarla de acuerdo a las necesidades de cada subsistema y aplicación.

6. Debemos aclarar todos aquellos aspectos que no queden claros en el diagrama, usando notas.

7. En general, debemos tener en cuenta que el diseño es una evolución del análisis, por lo que las clases que incluyamos en el diseño deben corresponderse razonablemente con todo el trabajo hecho en esa parte del documento.

6.2.1. Diagrama de Clases

Es necesario mostrar por lo menos un diagrama global de clases. Si el diagrama de clases global fuese muy grande, podemos dividir el diagrama en varios diagramas más pequeños según la estructura de la aplicación, mostrando el principal abreviado tal y como se explico anteriormente en los diagramas de paquetes. Otra opción es hacer que la hoja que tenga el diagrama se imprima en otro formato que nos proporcione más espacio (en horizontal, A3,...).

6.3. Diagramas de Interacción y Estados

Esta sección se usará, entre otras cosas, para evolucionar y detallar los diagramas de robustez que hemos desarrollado en el análisis usando diagramas de interacción y de estados. La estructura a seguir es la de incluir el diagrama en sí (dibujo) y luego hacer una lista explicando cada uno de los pasos existentes en dicho diagrama que lo requieran. Los diagramas deben incluir nombres de clases, métodos y parámetros "reales", con la intención de que puedan trasladarse directamente a la implementación del sistema (el objetivo de un buen diseño). Por este motivo no debemos escatimar en detalles a la hora de desarrollarlos, ya que se supone que de estos diagramas podremos extraer directamente la implementación de la aplicación.

Cuando se menciona que los diagramas del diseño son una evolución de los del análisis significa que podemos aprovechar el trabajo ya realizado para su creación, adaptando la información en los diagramas correspondientes del análisis para desarrollar los nuevos diagramas (se debe lograr coherencia entre las entidades desarrolladas en ambas fases).

En cuanto al número de diagramas a contemplar, al igual que se dijo anteriormente no es necesario incluir en el diseño todos y cada uno de los posibles diagramas, sino que podremos ahorrar extender aquellos en los que:

1. Las operaciones sean muy simples (en cuyo caso se puede optar por no decir nada al respecto o colocar simplemente un texto explicativo).

2. La operación sea muy similar o idéntica a una que ya está desarrollada en esta sección. En este caso lo mejor es poner en el diagrama original a que otros casos se aplica o representa y no repetirlo.

6.3.1. Caso de Uso 1.1

Diagramas de Interacción (Comunicación y Secuencia)

Diagramas de secuencia de diseño correspondiente al escenario principal del caso de uso y a aquellos alternativos que merezcan ser desarrollados (estableciendo un paralelismo con la nomenclatura usada en el análisis). Podemos encontrar más información en: `pst.web.cern.ch/PST/HandBookWorkBook/Handbook/SoftwareEngineering/UCDOM_interaction.html`

En el caso de que creamos necesario mostrar el comportamiento de varios objetos que colaboran para la consecución de un determinado fin común, debemos crear un diagrama de comunicación de objetos que represente este comportamiento. Podemos encontrar más información acerca de los mismos en esta dirección: `http://www.agilemodeling.com/artifacts/communicationDiagram.htm`

Diagramas de Estados de las Clases

En caso necesario, si es conveniente dar más detalles de cómo se comporta un determinado escenario, podemos incluir diagramas de estados de las clases involucradas en el mismo para indicar las distintas fases por las que una clase pasará durante el mismo. Podemos encontrar más información sobre estos diagramas en: `http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm`

6.3.2. Caso de Uso 1.2

...

6.4. Diagramas de Actividades

Si hay alguna operación o funcionalidad dentro del sistema que merezca la pena destacar se documentará mediante un diagrama de actividades. Puede haber tantos como consideréis necesarios. Más información acerca de estos diagramas se puede encontrar en estos enlaces:

- `http://dn.codegear.com/article/31863#activity-diagrams`
- `http://www.agilemodeling.com/artifacts/activityDiagram.htm`

6.5. Diseño de la Base de Datos

Esta sección es la indicada para describir todo lo relativo al sistema de gestión de bases de datos que vamos a usar en nuestra aplicación (si usamos alguno). Esto incluye hablar del sistema en sí, la versión utilizada, las clases empleadas al usarlo, como se integra en el sistema y el diagrama E-R de las tablas y relaciones creadas, todo ello en los apartados que se muestran a continuación.

6.5.1. Descripción del SGBD Usado

...

6.5.2. Integración del SGBD en Nuestro Sistema

...

6.5.3. Diagrama E-R

Imagen o imagenes del diagrama entidad-relación de la BBDD usada.

6.6. Diseño de la Interfaz

Esta sección debe mostrar ya la interfaz definitiva de la aplicación (que evidentemente deberá ser una evolución del diseño mostrado en el análisis) y las diferentes partes de las que consta. Como es más que probable que todas las pantallas tengan elementos de interfaz comunes, esta es la sección donde se va a hablar de cada uno de esos elementos, su propósito y su función (barra de menús, barras de estado, etc.).

En caso de no haber hecho un prototipo realista en la fase de análisis (es decir, mostrado el aspecto real del programa), debemos hacerlo aquí. En todo caso, cualquier cosa que quedase pendiente en la fase de análisis acerca de la descripción del interfaz debe concretarse aquí.

6.7. Especificación Técnica del Plan de Pruebas

El proceso de pruebas se extiende durante todo el proceso de construcción del software y por tanto en esta sección debemos describir cómo vamos a aplicar las pruebas diseñadas y especificar más otro tipo de pruebas que vamos a realizar al software.

No debemos olvidarnos mencionar entonces bajo que máquina (incluyendo sistema operativo y entorno de desarrollo) se realizan las pruebas, y en qué orden se realizan las diferentes pruebas contempladas, entre otros aspectos.

NOTA: Es perfectamente posible que, aunque un sistema conste de múltiples procesos independientes que colaboran entre sí (por ejemplo un cliente y un servidor), las pruebas se hagan en sólo una máquina (Ej.: cliente y servidor que se conectan a *localhost*).

6.7.1. Pruebas Unitarias

Esta sección contendrá la descripción de la aplicación de las pruebas unitarias que hemos descrito anteriormente (qué datos se van a introducir finalmente sobre qué clases, ahora que conocemos finalmente todas las clases que se van a implementar gracias al diseño), indicando cuando las vamos a realizar y que medidas tomaremos en caso de encontrar fallos. También debemos hablar de si hemos dividido las pruebas entre los diferentes subsistemas de alguna forma.

En este apartado debemos también describir las posibles **pruebas de regresión** que queramos implementar. Estas pruebas de regresión se deben automatizar para poder pasarlas periódicamente cada vez que hagamos cambios importantes en el sistema.

Se recomienda encarecidamente el empleo de una herramienta de automatización de estas pruebas unitarias para facilitar mucho esta labor. Posibles herramientas son *JUnit* (Java) o *NUnit*. Realmente esta sería la forma más adecuada de afrontar esta tarea.

6.7.2. Pruebas de Integración y del Sistema

Las pruebas funcionales y del sistema que hemos especificado en el análisis deben aplicarse para garantizar que el sistema funciona correctamente. Debemos describir así cómo y cuándo vamos a aplicar este tipo de casos de prueba dentro del sistema, así como las entradas y salidas de estas pruebas. No debemos olvidar que la aplicación de las pruebas y qué ocurre cuando se pasan irá en una sección posterior.

Debemos recordar pues que, tanto en esta sección como en la anterior, ya tenemos las pruebas diseñadas en la fase de análisis, y que ahora se trataría de describir cómo y cuando las vamos a aplicar, qué datos vamos a introducir en cada caso y qué vamos a hacer en caso de acierto o fallo de las pruebas. No se trata de repetir lo mismo de nuevo.

A modo de guía, debemos pensar que lo que diseñemos aquí debe ser directamente aplicable sobre el código de la aplicación.

6.7.3. Pruebas de Usabilidad y Accesibilidad

Dado que es posible que el lector de este documento no esté familiarizado con este tipo de pruebas, haremos una descripción un poco más en detalle de en qué consisten. Este tipo de pruebas determinan la satisfacción del cliente con el producto final y por tanto debemos darle mucha importancia a las mismas, no dejándolas de lado en ningún caso.

Las pruebas de usabilidad tratan de evaluar la aplicación en su funcionamiento real. Debemos realizar un conjunto de pruebas para verificar que las distintas partes del programa se pueden usar adecuadamente. Los objetivos son:

1. Mejorar la calidad de la interacción de los usuarios con nuestra aplicación.
2. Reducir el tiempo necesario para hacer las distintas tareas en la aplicación y de esta forma aumentar la productividad.

Debemos tener los siguientes elementos en cuenta, para describirlos completamente en esta sección antes de hacer las pruebas en sí:

1. **Usuarios**: Distintos grupos de usuarios de la aplicación sobre los que vamos a realizar las pruebas. Cada usuario tendrá un perfil de uso distinto y tenemos que tenerlo en cuenta a la hora de hacer las pruebas. Por lo general es suficiente con efectuar las pruebas con un número reducido de usuarios (de 3 a 5). También debemos dejar bien claro, en caso de que los usuarios lo deseen, la profesión y la edad de los mismos.

2. **Lugar de realización**: En un laboratorio o bien en la propia casa u oficina del cliente.

3. **Metodología**: Describir que vamos a hacer en estas pruebas (el conjunto de pasos a seguir en las mismas).

Aunque en la siguiente sección se dan detalles acerca de cómo hacer cuestionarios para efectuar estas pruebas, estos no son la única herramienta para ello. Existen otros tipos de herramientas como pruebas basadas en las opiniones dadas de forma oral por los usuarios, heurísticas, etc. Dependiendo fundamentalmente del producto y para quien este destinado. No obstante, los cuestionarios porque son una herramienta adecuada para gran parte de los casos y no exigen muchos recursos. Se recomienda consultar al director de proyecto acerca de la mejor forma de hacer este tipo de pruebas en el proyecto concreto a desarrollar.

Diseño de Cuestionarios

Detallamos a continuación algunas pautas a seguir en el diseño de los cuestionarios que podamos necesitar, si se estima oportuno el uso de esta herramienta. Una norma muy importante a cumplir es que él tiempo necesario para rellenar los cuestionarios no deben superar los 15 minutos.

Cuestionario de Evaluación

Debemos elaborar un cuestionario para que los usuarios lo hagan y determinar así distintos aspectos del mismo y de su interacción con la aplicación. Los puntos a tocar son (esto es un esquema que puede ampliarse si se desea para adaptarlo a la aplicación en sí):

1. **Preguntas de carácter general** a través de las cuales se determine el tipo de usuario y su nivel de conocimiento informático.

2. **Actividades guiadas** para hacer con nuestra aplicación.

3. **Batería de preguntas cortas** con los distintos aspectos de la aplicación que se pretendan evaluar.

4. **Observaciones**, para que el usuario aporte todo lo que considere oportuno de nuestra aplicación.

Cuestionario para el Responsable de las Pruebas

Además, debemos desarrollar un cuestionario para que el responsable de las pruebas pueda anotar distintos aspectos que observe durante la realización de las pruebas. Un posible desarrollo de todos estos aspectos se describe a continuación.

Actividades de las Pruebas de Usabilidad

Preguntas de carácter general

Se muestra un esbozo de un posible cuestionario, que debemos desarrollar y adaptar a nuestras necesidades:

Preguntas de carácter general
¿Usa un ordenador frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Qué tipo de actividades realiza con el ordenador?
1. Es parte de mi trabajo o profesión 2. Lo uso básicamente para ocio 3. Solo empleo aplicaciones estilo Office 4. Únicamente leo el correo y navego ocasionalmente
¿Ha usado alguna vez software como el de esta prueba?
1. Sí, he empleado software similar 2. No, aunque si empleo otros programas que me ayudan a realizar tareas similares 3. No, nunca
¿Qué busca Vd. Principalmente en un programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido 4. Que tenga todas las funciones necesarias

Actividades guiadas

Hacer un compendio de actividades que se puedan hacer con la aplicación para que nuestros usuarios las hagan y comenten los problemas y dificultades que según su opinión presenta la aplicación (si existiesen). Debemos recoger estas opiniones en el documento. Posibles actividades a probar son:

1. Añadir un usuario a la aplicación
2. Eliminar un artículo de la aplicación
3. ...

Preguntas Cortas sobre la Aplicación y Observaciones

Un posible cuestionario de preguntas cortas (a desarrollar más en cada proyecto) es el siguiente:

Preguntas Cortas sobre la Aplicación y Observaciones				
Facilidad de Uso	**Siempre**	**Muy Frecuente**	**Poco Frecuente**	**Nunca**
¿Sabe donde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le resulta sencillo el uso de la aplicación?				
Funcionalidad	**Siempre**	**Muy Frecuente**	**Poco Frecuente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				
Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**No Adecuado**
El tipo y tamaño de letra es				
Los iconos e imágenes usados son				
Los colores empleados son				
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?				
¿El diseño de las pantallas es claro y atractivo?				
¿Cree que el programa está bien estructurado?				
Observaciones				
Cualquier comentario del usuario				

Evidentemente debemos extender este cuestionario de ejemplo para adaptarlo a nues-

tras necesidades.

Cuestionario para el Responsable de las Pruebas

Cuestionario para el Responsable de las Pruebas	
Aspecto Observado	**Notas**
El usuario comienza a trabajar de forma rápida por las tareas	
Tiempo en realizar cada tarea	
Errores leves cometidos	
Errores graves cometidos	
<Incluir otras cosas>	
...	
...	

Los resultados de las pruebas se deben analizar conjuntamente para así establecer una conclusión respecto a cuatro aspectos de usabilidad:

1. **Facilidad de aprendizaje**: Capacidad para aprender la funcionalidad de la aplicación desarrollada y desarrollar las tareas de manera adecuada, midiendo cuanto se tarda en hacer las distintas tareas.

2. **Eficiencia**: Cuanto mejora la labor de los usuarios por usar la aplicación respecto a lo que se hacía anteriormente.

3. **Errores**: Cuantos errores cometen los usuarios en las distintas tareas, lo que decrementa la usabilidad del mismo.

4. **Satisfacción del usuario**: Impresión general de los usuarios al usar la aplicación.

Pruebas de Accesibilidad

El grueso de las **pruebas de accesibilidad** será descrito en la parte de desarrollo de las pruebas. A nivel de diseño simplemente puede indicarse qué tipo de pruebas van a realizarse para el programa concreto, que normas *WCAG* van a seguirse u otras consideraciones generales acerca de las mismas, dentro de las posibles vías de actuación. *WAI* propone básicamente tres tipos de revisiones:

1. **Revisión preliminar**: Identifica rápidamente problemas de accesibilidad de un sitio *Web*. Todos los procesos de esta revisión también tienen lugar en la siguiente revisión.

2. **Evaluación de conformidad**: Permite indicar si un sitio web cumple con los estándares de accesibilidad (Ej.: *WCAG*). Puede encontrarse más información en: `www.w3.org/WAI/eval`, aunque en esta plantilla se intentarán dar pautas para hacer una evaluación adecuada.

3. **Otras evaluaciones**: *WAI* también proporciona otra serie de documentos para evaluar aplicaciones en contextos específicos y para implicar a usuarios discapacitados en la evaluación. En el primer caso se tratan de documentos complementarios a los anteriores que describen consideraciones para la evaluación de sitios grandes y complejos durante el proceso de desarrollo y mantenimiento, o bien para sitios existentes cuyas páginas son generadas dinámicamente.

6.7.4. Pruebas de Rendimiento

El sistema desarrollado consumirá una determinada cantidad de recursos (memoria y tiempo de proceso) que debemos procurar que sean los menores posible. Por ello, la aplicación debe al menos medirse para ver cuántos recursos consume y se debe intentar eliminar posibles cuellos de botella en el rendimiento que se puedan presentar en uno o varios subsistemas de la misma. Debemos diseñar medios para hacer estas mediciones en nuestra aplicación.

Para esta tarea debemos pues medir el tiempo que tardan las operaciones más importantes que haga el sistema y comprobar si es posible mejorarlas de algún modo una vez que el sistema esté en funcionamiento. En este apartado debemos diseñar que actuaciones vamos a llevar a cabo para hacer estas pruebas y las actuaciones que vamos a llevar a cabo para mejorar el rendimiento en aquellos puntos en los que encontremos problemas. Una herramienta que podemos aplicar para este fin es la opción "*Tools – View Speed Report*" de las "*Web Developer Extensions*" de *Firefox*.

Una consideración a tener en cuenta es determinar qué actividades serán las más frecuentes y si su rendimiento es adecuado o no, ya que por esta vía conseguiremos una mejor optimización de la aplicación.

En el caso de que el programa deba tener algún requisito respecto al tiempo que debe tardar o a la memoria ocupada, debe tenerse especial cuidado en este aspecto.

Por último, también debemos diseñar pruebas de carga para determinar cómo responde el sistema con un alto número de usuarios o un gran volumen de datos. Para esta tarea puede ayudar una aplicación como *JMeter* (`http://jakarta.apache.org/jmeter/`).

Capítulo 7

Implementación del Sistema

7.1. Estándares y Normas Seguidos

Descripción breve de los estándares y normas que hayamos usado en nuestra aplicación a la hora de desarrollar su código y si nos hemos ocupado de validar que esos estándares se cumplan efectivamente.

7.2. Lenguajes de Programación

Descripción de los lenguajes de programación usados, su versión y distribución, los módulos o complementos de los mismos empleados, etc.

7.3. Herramientas y Programas Usados para el Desarrollo

Descripción de todas las herramientas de desarrollo (Eclipse, Visual Studio, etc.), sistemas adicionales existentes, complementos y otros productos software que necesitemos para la implementación de nuestro sistema. Debemos dejar claro que versión usamos, para qué y cómo interactuará con nuestro sistema.

7.3.1. Programa 1

...

7.3.2. Programa 2

...

7.4. Creación del Sistema

Todos los aspectos con los que nos hemos encontrado durante la implementación debemos describirlos aquí.

7.4.1. Problemas Encontrados

Enumeramos los problemas encontradosproblemas encontrados en el desarrollo y la solución que le hemos dado, si hubo alguno que merezca la pena destacar.

7.4.2. Descripción Detallada de las Clases

En esta sección debemos describir las clases más relevantes de nuestra aplicación de manera que se detallen todos sus atributos y métodos así como su descripción y función exacta, tal y como aparecen en el código de la aplicación. Para hacerlo lo mejor posible debemos elaborar esta descripción una vez cerrado el desarrollo de la aplicación en sí, para evitar problemas de versiones por revisiones al código que surjan durante el desarrollo. Para ello podemos hacer dos cosas:

1. Emplear una herramienta que vuelque las clases y sus comentarios a un formato que podamos usar en este documento y que contemple todos los aspectos vistos antes (es decir, que el resultado sea similar a la tabla mostrada más abajo). Herramientas como *Javadoc* o similares son adecuadas para este fin.

2. En caso de no disponer o no querer usar una herramienta como la mencionada anteriormente, empleamos la tabla que se muestra a continuación.

Para cada clase se pondrá un título de nivel 4 y se comenzará en una página nueva. NO rellenar con saltos de párrafo (*return*).

Clase 1:

Descripcion detallada de una clase			
Nombre	**Tipo**	**Descripción**	**Hereda de...**
	abstract, final, ...	¿Para qué sirve?	
Responsabilidades			
Número	**Descripción**		
Métodos			
Acceso y/o Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
(public, protected, private) (abstract, virtual, static)			

continua en la próxima página

Descripcion detallada de una clase			
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
(public, protected, private)	*static, final*		
Observaciones			
Cualquier comentario adicional a la clase...			

En caso de ser un proyecto hecho en parejas, conviene indicar adicionalmente en estas tablas quien ha sido el autor de esta clase.

Capítulo 8

Desarrollo de las Pruebas

8.1. Pruebas Unitarias

Describir aquí el resultado de las pruebas unitarias que ya hemos diseñado y descrito anteriormente, así como los resultados obtenidos y las actuaciones que hemos llevado a cabo ante los errores y problemas encontrados.

8.2. Pruebas de Integración y del Sistema

Ejecutamos las pruebas funcionales ya diseñadas anteriormente y anotamos el resultado que obtenemos, comparándolo con el que especificamos anteriormente. Podemos hacerlo a partir de una tabla modificada de la anterior como ésta (si es más sencillo, puede hacerse con pequeñas tablas independientes para cada caso):

Caso de Uso 1.1: Añadir Usuario	
Entrada	Resultado Esperado
Añadir un usuario no existente	El sistema posee un usuario más
	Resultado Obtenido
	El sistema efectivamente posee un usuario más
Entrada	Resultado Esperado
Añadir un usuario que ya existe	El sistema no posee un usuario más y se muestra un dialogo notificándolo
	Resultado Obtenido
Entrada	Resultado Esperado
Cancelar la Operación	El sistema permanece sin cambios.
	Resultado Obtenido

8.3. Pruebas de Usabilidad y Accesibilidad

8.3.1. Pruebas de Usabilidad

A partir de los cuestionarios que se diseñaron anteriormente y de los procedimientos explicados, mostramos aquí el resultado de aplicarlos sobre todos los usuarios que hayamos empleado para estas pruebas.

Es muy importante a su vez indicar los cambios producidos en la interfaz a partir de las sugerencias recogidas por los usuarios. Una forma adecuada de representar esto sería poner la pantalla inicial y poner la pantalla después de los cambios efectuados.

Además de los cuestionarios diseñados anteriormente, podemos pasar esta guía de usabilidad desarrollada por Yusef Hassan Montero [6] (`http://www.nosolousabilidad.com/articulos/heuristica.htm`):

Criterios	**¿Se cumple?**
Generales	
¿Cuáles son los objetivos del sitio web? ¿Son concretos y bien definidos? ¿Los contenidos y servicios que ofrece se corresponden con esos objetivos?	**SI**
¿Tiene una URL correcta, clara y fácil de recordar? ¿Y las URL de sus páginas internas? ¿Son claras y permanentes?	
¿Muestra de forma precisa y completa qué contenidos o servicios ofrece realmente el sitio web? El diseño de la página de inicio debe ser diferente al resto de páginas y cumplir la función de 'escaparate' del sitio.	
¿La estructura general del sitio web está orientada al usuario? Los sitios web deben estructurarse pensando en el usuario, sus objetivos y necesidades. La estructura interna de la empresa u organización, cómo funciona o se organiza no interesan al usuario.	
¿El *look & feel* general se corresponde con los objetivos, características, contenidos y servicios del sitio web? Ciertas combinaciones de colores ofrecen imágenes más o menos formales, serias o profesionales.	
¿Es coherente el diseño general del sitio web? Se debe mantener una coherencia y uniformidad en las estructuras y colores de todas las páginas. Esto sirve para que el usuario no se desoriente en su navegación.	
¿Es reconocible el diseño general del sitio web? Cuánto más se parezca el sitio web al resto de sitios web, más fácil será de usar.	
¿El sitio web se actualiza periódicamente? ¿Indica cuándo se actualiza? Las fechas que se muestren en la página deben corresponderse con actualizaciones, noticias, eventos...no con la fecha del sistema del usuario.	
Identidad e Información	

continua en la próxima página

Criterios	¿Se cumple?
¿Se muestra claramente la identidad de la empresa-sitio a través de todas las páginas?	
El Logotipo, **¿es significativo, identificable y suficientemente visible?**	
El eslogan o *tagline*, **¿expresa realmente qué es la empresa y qué servicios ofrece?**	
¿Se ofrece algún enlace con información sobre la empresa, sitio web, 'webmaster',...?	
¿Se proporciona mecanismos para ponerse en contacto con la empresa? (email, teléfono, dirección postal, fax...)	
¿Se proporciona información sobre la protección de datos de carácter personal de los clientes o los derechos de autor de los contenidos del sitio web?	
En artículos, noticias, informes... **¿Se muestra claramente información sobre el autor, fuentes y fechas de creación y revisión del documento?**	
Lenguaje y Redacción	
¿El sitio web habla el mismo lenguaje que sus usuarios? Se debe evitar usar un lenguaje corporativista. Así mismo, hay que prestarle especial atención al idioma, y ofrecer versiones del sitio en diferentes idiomas cuando sea necesario.	
¿Emplea un lenguaje claro y conciso?	
¿Es amigable, familiar y cercano? Es decir, lo contrario a utilizar un lenguaje constantemente imperativo, mensajes crípticos, o tratar con "desprecio" al usuario.	
¿1 párrafo = 1 idea? Cada párrafo es un objeto informativo. Trasmita ideas, mensajes...Se deben evitar párrafos vacíos o varios mensajes en un mismo párrafo.	
Rotulado	
Los rótulos, **¿son significativos?** Ejemplo: evitar rótulos del tipo "haga clic aquí".	
¿Usa rótulos estándar? Siempre que exista un "estándar" aceptado comúnmente para el caso concreto, como "Mapa del Sitio" o "Acerca de..."	
¿Usa un único sistema de organización, bien definido y claro? No se deben mezclar diferentes. Los sistemas de organización son: alfabético, geográfico, cronológico, temático, orientado a tareas, orientado al público y orientado a metáforas.	

continua en la próxima página

Criterios	**¿Se cumple?**
¿Utiliza un sistema de rotulado controlado y preciso? Por ejemplo, si un enlace tiene el rótulo "Quiénes somos", no puede dirigir a una página cuyo encabezamiento sea "Acerca de"	
El título de las páginas, **¿Es correcto? ¿Ha sido planificado?** Relacionado con la capacidad para poder buscar y encontrar el sitio *web*.	
Estructura y Navegación	
La estructura de organización y navegación, **¿Es la más adecuada?** Hay varios tipos de estructuras: jerárquicas, hipertextual, facetada,...	
En el caso de estructura jerárquica, **¿Mantiene un equilibrio entre Profundidad y Anchura?**	
En el caso de ser puramente hipertextual, **¿Están todos los clúster de nodos comunicados?** Aquí se mide la distancia entre nodos.	
¿Los enlaces son fácilmente reconocibles como tales? ¿Su caracterización indica su estado (visitados, activos,...)? Los enlaces no sólo deben reconocerse como tales, sino que su caracterización debe indicar su estado, y ser reconocidos como una unidad	
En menús de navegación, **¿Se ha controlado el número de elementos y de términos por elemento para no producir sobrecarga memorística?** No se deben superar los 7±2 elementos, ni los 2 o, como mucho, 3 términos por elemento.	
¿Es predecible la respuesta del sistema antes de hacer clic sobre el enlace? Relacionado con el nivel de significación del rótulo del enlace, aunque también con: el uso de globos de texto, información contextual, la barra de estado del navegador,...	
¿Se ha controlado que no haya enlaces que no llevan a ningún sitio? Enlaces que no llevan a ningún sitio: Los enlaces rotos, y los que enlazan con la misma página que se está visualizando (por ejemplo enlaces a la "home" desde la misma página de inicio)	
¿Existen elementos de navegación que orienten al usuario acerca de dónde está y cómo deshacer su navegación? ...como *breadcrumbs*, enlaces a la página de inicio,...recuerde que el logo también es recomendable que enlace con la página de inicio.	
Las imágenes enlace, **¿se reconocen como clicables? ¿Incluyen un atributo 'title' describiendo la página de destino?** En este sentido, también hay que cuidar que no haya imágenes que parezcan enlaces y en realidad no lo sean.	
¿Se ha evitado la redundancia de enlaces?	
¿Se ha controlado que no haya páginas "huérfanas"? Páginas huérfanas: que aún siendo enlazadas desde otras páginas, éstas no enlacen con ninguna.	
Layout de la Página	

continua en la próxima página

Criterios	¿Se cumple?
¿Se aprovechan las zonas de alta jerarquía informativa de la página para contenidos de mayor relevancia? (como por ejemplo la zona central)	
¿Se ha evitado la sobrecarga informativa? Esto se consigue haciendo un uso correcto de colores, efectos tipográficos y agrupaciones para discriminar información. Los grupos diferentes de objetos informativos de una página deben ser 7±2.	
¿Es una interfaz limpia, sin ruido visual?	
¿Existen zonas en "blanco" entre los objetos informativos de la página para poder descansar la vista?	
¿Se hace un uso correcto del espacio visual de la página? Es decir, que no se desaproveche demasiado espacio con elementos de decoración, o grandes zonas en "blanco", y que no se adjudique demasiado espacio a elementos de menor importancia.	
¿Se utiliza correctamente la jerarquía visual para expresar las relaciones del tipo "parte de" entre los elementos de la página? (La jerarquía visual se utiliza para orientar al usuario)	
¿Se ha controlado la longitud de página? Se debe evitar en la medida de lo posible el *scrolling*. Si la página es muy extensa, se debe fraccionar.	
Búsqueda (si es necesario por su extensión añadir un buscador interno al sitio)	
¿Se encuentra fácilmente accesible? Es decir: directamente desde la home, y a ser posible desde todas las páginas del sitio, y colocado en la zona superior de la página.	
¿Es fácilmente reconocible como tal?	
¿Permite la búsqueda avanzada? (siempre y cuando, por las características del sitio web, fuera de utilidad que la ofreciera)	
¿Muestra los resultados de la búsqueda de forma comprensible para el usuario?	
¿La caja de texto es lo suficientemente ancha?	
¿Asiste al usuario en caso de no poder ofrecer resultados para una consultada dada?	
Elementos Multimedia	
¿Las fotografías están bien recortadas? ¿Son comprensibles? ¿Se ha cuidado su resolución?	
¿Las metáforas visuales son reconocibles y comprensibles por cualquier usuario? (prestar especial atención a usuarios de otros países y culturas)	
¿El uso de imágenes o animaciones proporciona algún tipo de valor añadido?	

continua en la próxima página

Criterios	**¿Se cumple?**
¿Se ha evitado el uso de animaciones cíclicas?	
Ayuda	
Si posee una sección de Ayuda, **¿Es verdaderamente necesaria?** Siempre que se pueda prescindir de ella simplificando los elementos de navegación e interacción, debe omitirse esta sección.	
En enlace a la sección de Ayuda, **¿Está colocado en una zona visible y "estándar"?** La zona de la página más normal para incluir el enlace a la sección de Ayuda, es la superior derecha.	
¿Se ofrece ayuda contextual en tareas complejas? (transferencias bancarias, formularios de registro...)	
Si posee *FAQs*, **¿Es correcta tanto la elección como la redacción de las preguntas? ¿Y las respuestas?**	
Accesibilidad (debería cubrirse con los test de Accesibilidad posteriores)	
¿El tamaño de fuente se ha definido de forma relativa, o por lo menos, la fuente es lo suficientemente grande como para no dificultar la legibilidad del texto?	
¿El tipo de fuente, efectos tipográficos, ancho de línea y alineación empleadas facilitan la lectura?	
¿Existe un alto contraste entre el color de fuente y el fondo?	
¿Incluyen las imágenes atributos 'alt' que describan su contenido?	
¿Es compatible el sitio web con los diferentes navegadores? ¿Se visualiza correctamente con diferentes resoluciones de pantalla? Se debe prestar atención a: *JScript*, *CSS*, tablas, fuentes...	
¿Puede el usuario disfrutar de todos los contenidos del sitio web sin necesidad de tener que descargar e instalar *plugins* adicionales?	
¿Se ha controlado el peso de la página? Se deben optimizar las imágenes, controlar el tamaño del código *JScript*...	
¿Se puede imprimir la página sin problemas? Leer en pantalla es molesto, por lo que muchos usuarios preferirán imprimir las páginas para leerlas. Se debe asegurar que se puede imprimir la página (no salen partes cortadas), y que el resultado es legible.	
Control y Retroalimentación	
¿Tiene el usuario todo el control sobre el interfaz? Se debe evitar el uso de ventanas pop-up, ventanas que se abren a pantalla completa, banners intrusivos...	

continua en la próxima página

Criterios	**¿Se cumple?**
¿Se informa constantemente al usuario acerca de lo que está pasando? Si el usuario tiene que esperar hasta que se termine una operación, se debe mostrar un mensaje indicándoselo y que debe esperar, con el tiempo de espera estimado o una barra de progreso.	
¿Se informa al usuario de lo que ha pasado? Por ejemplo, cuando un usuario valora un artículo o responde a una encuesta, se le debe informar de que su voto ha sido procesado correctamente.	
Cuando se produce un error, **¿se informa de forma clara y no alarmista al usuario de lo ocurrido y de cómo solucionar el problema?** Siempre es mejor intentar evitar que se produzcan errores a tener que informar al usuario del error.	
¿Posee el usuario libertad para actuar? NO restringir la libertad del usuario: Uso de animaciones que no pueden ser "saltadas", páginas en las que desaparecen los botones de navegación, no impida al usuario poder usar el botón derecho de su ratón...	
¿Se ha controlado el tiempo de respuesta? Esto tiene que ver con el peso de cada página (accesibilidad) y tiene relación con el tiempo que tarda el servidor en finalizar una tarea y responder. El tiempo máximo que esperará un usuario son 10 segundos	
Aclaraciones	
¿Se ha evaluado adecuadamente la orientación del usuario? (Donde estoy, como volver, que he visitado, que va a pasar) ([1])	
¿Se ha usado correctamente la publicidad? ([2])	

Tras rellenar esta tabla se deben poner a continuación una enumeración de todas aquellas aclaraciones y/o observaciones que queramos hacer acerca de la misma.

8.3.2. Pruebas de Accesibilidad

A continuación se detallan las tareas que se recomiendan hacer para asegurarnos de que el programa creado cumple con los estándares de accesibilidad. Esta sección está muy enfocada a proyectos web, pero algunas ideas pueden extrapolarse a programas de escritorio en caso necesario. Un proyecto debería hacer una evaluación de conformidad completa del mismo, pero se ha incluido un procedimiento más abreviado (revisión preliminar) para aquellos casos que por alguna razón se necesite pasar un procedimiento más rápido o menos exhaustivo. Una forma de proceder es hacer la revisión preliminar y, una vez superada arreglando todos los defectos encontrados, intentar ir a por la evaluación de conformidad completa, que permitirá reutilizar todo el estudio hecho en la revisión preliminar para hacerla.

[1] La orientación del usuario se puede evaluar a través de varios criterios: elementos de navegación orientativos, caracterización de los enlaces e información contextual en elementos de interacción (estructura y navegación); distribución visual de la página (layout); coherencia del diseño (generales); nivel de significación de los rótulos (rotulado) y retroalimentación del usuarios (control y retroalimentación)

[2] Respecto a la publicidad (normalmente en forma de banners), se puede evaluar desde varios criterios: lenguaje (lenguaje y redacción), nivel de significación de los rótulos (rotulado), jerarquía informativa y ruido visual (layout de la página), pop-ups y banners intrusivos (control y retroalimentación)...

Revisión Preliminar

Como paso previo a hacer los pasos descritos, es necesario enumerar el material utilizado para ello, como navegadores (*IE*, *Firefox*,...), lectores de pantalla o cualquier otra herramienta usada para hacer las pruebas.

Paso 1. Selección de un grupo de páginas representativo de la aplicación

Para pasar las pruebas de accesibilidad es necesario escoger una muestra de páginas de la aplicación representativa para así no tener que someter a toda la aplicación al proceso de revisión. Una muestra representativa está formada por:

1. La página de entrada al sitio / principal / home
2. Páginas con distinta organización y funcionalidad, como por ejemplo:
   a) Páginas con tablas
   b) Páginas con formularios
   c) Páginas con diagramas o gráficos
   d) Páginas con scripts o aplicaciones
   e) Páginas con resultados generados dinámicamente (por ejemplo, con un gestor de contenidos)

Paso 2. Examinar las páginas usando un navegador gráfico

Esta actividad comprende las siguientes operaciones:

1. Desactivar las imágenes y probar como queda el aspecto de la aplicación. Se puede hacer fácilmente mediante el menú "*Images*" de la "*Web Developer Extension*" de *Firefox* de la que se habló anteriormente.
2. Apagar los altavoces (si la página tuviese alguna clase de narración oral o sonido)
3. Cambiar el tamaño del texto y comprobar que sigue siendo usable (muchos navegadores permiten hacer esto fácilmente con *CTRL + Rueda del ratón*).
4. Cambiar solamente el tamaño de letra de la página para ver cómo se comporta. En *Firefox* podemos ir a "Herramientas – Opciones – Contenido", aunque también podemos hacerlo cambiando la *CSS* de la propia página si el navegador no soporta esta opción. Si un tamaño de letra estándar es 16, se puede probar con un tamaño mínimo (9) y un máximo (72) para ver qué ocurre. Posteriormente podemos hacer pruebas con dos tamaños intermedios (32 y 48), para ver qué ocurriría con nuestra página si los usuarios necesitan un tamaño de letra muy superior al estándar.
5. Cambiar la resolución y el tamaño de la ventana (verificando que no es necesario el *scroll* horizontal). Esto puede hacerse fácilmente con el *plugin* "*Web Developer Extension*" de *Firefox*. Este *plugin*, una vez instalado tiene una opción "*Resize*". Mediante su sub-opción "*Edit Resize Dimensions*" podemos introducir nuevas resoluciones que podemos usar para comprobar cómo se comporta nuestra aplicación ante ellas. Resoluciones interesantes a probar son: 800x600, 1024x768, 1152x864, 1280x720, 1280x768, 1280x960, 1280x1024 y 1600x1200. Las resoluciones a probar están condicionadas por la máxima resolución de nuestro monitor, por lo que se recomienda poner el mismo a la máxima resolución posible antes de hacer estas pruebas.

6. Relacionado con lo anterior, también se recomienda, si es posible, comprobar cómo se ve nuestra aplicación en múltiples tamaños de pantalla (15 pulgadas, 17 pulgadas,...). Otra posible prueba es redimensionar la página múltiples veces (probando a hacerla cada vez más grande y cada vez más pequeña) para ver cómo se comporta su contenido ante esta situación. Si el aspecto de la página se estropea por ello, entonces habremos encontrado un problema.

7. Probar a visualizar la página sin sus hojas de estilo *CSS* para asegurarse de que aún es legible y usable. Se puede hacer fácilmente mediante el menú "*CSS*" de la "*Web Developer Extension*" de *Firefox* de la que se habló anteriormente.

8. Probar a visualizar la página usando una escala de grises. Para esto podemos usar la herramienta web *GrayBit* (`http://graybit.com/main.php`).

9. Usar el teclado para navegar a través de los enlaces y controles de formularios, usando *Tab* para desplazarse.

Herramientas útiles para comprobar estos aspectos pueden ser:

1. *AIS Toolbar* para *Internet Explorer* y *Opera*

2. *WAVE Toolbar* para *Firefox*, *Internet Explorer* y *Netscape*

3. *Web Developer Extension* para *Firefox (*recomendada*)*

Paso 3. Examinar las páginas usando uno o varios navegadores especializados

Es muy necesario comprobar cómo nuestras páginas se comportan ante diferentes clases de navegadores, como por ejemplo:

1. Navegadores por voz como *Firevox* (`firevox.clcworld.net/`) (gratuito) o*JAWS* (`www.freedomscientific.com/jaws-hq.asp`). También puede encontrarse una lista de herramientas similares aquí: `en.wikipedia.org/wiki/Comparison_of_screen_readers`

2. Navegadores de texto como *Lynx*. Si se trabaja en *Windows* es posible ejecutarlo usando *Cygwin*.

En estos navegadores es necesario verificar que la información transmitida por ambos tipos es similar a la mostrada en el navegador gráfico y asegurarse de que el orden en el que se transmite dicha información es coherente.

Por otro lado, también es conveniente probar la página con diferentes navegadores (*IE*, *Firefox*, *Opera*, *Chrome*,...) y con distintas versiones de cada navegador. Aunque podría hacerse mediante *VMware*, instalar un gran nº de navegadores y versiones de los mismos es muy costoso. No obstante, una herramienta que nos proporcionará fácilmente acceso a un elevadísimo número de navegadores distintos, con múltiples versiones de cada uno y con distintos sistemas operativos, es *BrowserShots* (`http://browsershots.org/`). En esta página *Web* podremos introducir la *URL* de nuestro sitio (en caso de que no la tengamos disponible se puede probar con la *IP* de la máquina de desarrollo, asegurándose de que es accesible). Esto invocará a un servicio *web* que distribuirá la petición a múltiples máquinas con distintos navegadores instalados, devolviéndonos las capturas de pantalla en todos y cada uno de esos navegadores de nuestro sitio web, transcurrido un tiempo de proceso de la petición (en función de la hora del día y de la carga de trabajo del servicio puede tardar bastante). Si no cerramos la página o cancelamos la petición y tenemos paciencia, una vez terminado el proceso tendremos imágenes de nuestra web en multitud de navegadores y

SO diferentes fácilmente, algo que de otra manera nos supondría un coste muy elevado. Debido al coste en tiempo de esta herramienta, se recomienda hacerlo con la versión final del interfaz de la aplicación.

Paso 4. Utilizar herramientas automáticas de evaluación de accesibilidad

Este último paso consiste en pasar a la muestra de páginas seleccionadas herramientas de evaluación automática de la accesibilidad como:

1. *TAW* (`http://www.tawdis.net/taw3/cms/es`)
2. *HERA* (`http://www.sidar.org/hera/index.php.en`)
3. *EvalAccess* (`http://sipt07.si.ehu.es/evalaccess2/index.html`)
4. *WAVE* (`http://wave.webaim.org/`)

Debemos tener en cuenta:

1. Que hay que pasar un mínimo de dos herramientas de esta clase.
2. Que sólo pueden probar algunos aspectos de la accesibilidad. Una vez eliminados todos los errores de comprobación automática de la página, debemos hacer todo lo posible por eliminar todos los de comprobación manual (de todas las herramientas pasadas).

Paso 5. Resumen de resultados

Finalmente, se incluirá un pequeño informe final con los siguientes aspectos:

1. Tipos de problemas encontrados, como se han resuelto y aspectos positivos de la página.
2. Indicar como se detectó cada problema.

Evaluación de Conformidad

La evaluación de conformidad combina la evaluación manual de la página con la evaluación semiautomática. Se suele emplear cuando se desarrolla un sitio nuevo o bien cuando se evalúa un sitio existente. En el caso de un PFC, debemos intentar pasar una evaluación de conformidad completa al sitio para considerar que realmente ha hecho un esfuerzo adecuado por lograr un nivel de accesibilidad óptimo.

Paso 1. Determinar el alcance de la evaluación

Este paso contempla:

1. Definir y divulgar el nivel deseado de conformidad *WCAG 1.0* (*A*, *AA*, *AAA*). Se recuerda que las web destinadas a la administración pública deben tener al menos un nivel AA, siendo este un nivel mínimo exigible en un PFC. También pueden emplearse las normas *WCAG 2.0*. El siguiente enlace proporciona información de cómo pasar de la versión 1.0 a la versión 2.0 de forma resumida: `http://www.w3.org/WAI/WCAG20/from10/comparison/`
2. Seleccionar un conjunto representativo de páginas para la evaluación manual, siguiendo el criterio expuesto en la revisión preliminar.

Paso 2. Utilizar herramientas de evaluación automática de accesibilidad

Este paso contempla:

1. Validar los lenguajes utilizados (*HTML*, *CSS*,...) con las herramientas adecuadas que existan para ello. La "*Web Developer Extension*" de *Firefox* posee opciones para ello.

2. Usar al menos dos herramientas de evaluación automática en la muestra de páginas seleccionada (ver la revisión preliminar). También se debe usar al menos 1 herramienta en la totalidad del sitio.

3. Anotar y resolver los problemas encontrados.

Paso 3. Evaluar manualmente la muestra de páginas

Para ello debemos:

1. Utilizar el *checklist* de puntos de control del *WCAG* que aparecerá tras esta explicación, según versión. Como es mucho más probable que en los PFC se use la versión 1.0 de las normas, al final de esta sección se ha incluido este *checklist* adaptado a *Word* y listo para rellenarse:

   a) *WCAG 1.0*: `www.w3.org/TR/WCAG10/full-checklist.html`

   b) *WCAG 2.0*: `www.w3.org/TR/2006/WD-WCAG20-20060427/appendixB.html`

2. Examinar las páginas con al menos 3 navegadores gráficos en diferentes versiones y en diferentes plataformas (para lo cual puede usarse el servicio *web Browsershots* ya visto). Tener en cuenta especialmente estos puntos, que son una lista extendida de los que se enunciaron en la revisión preliminar. Para su comprobación podemos usar las mismas herramientas y procedimientos recomendados en dicha sección:

   a) Desactivar las imágenes y probar como queda el aspecto de la aplicación. Se puede hacer fácilmente mediante el menú "*Images*" de la "*Web Developer Extension*" de *Firefox* de la que se habló anteriormente.

   b) Apagar los altavoces (si la página tuviese alguna clase de narración oral o sonido)

   c) Cambiar el tamaño del texto y comprobar que sigue siendo usable (muchos navegadores permiten hacer esto fácilmente con *CTRL + Rueda del ratón*).

   d) Cambiar solamente el tamaño de letra de la página para ver cómo se comporta. En *Firefox* podemos ir a "Herramientas – Opciones – Contenido", aunque también podemos hacerlo cambiando la *CSS* de la propia página si el navegador no soporta esta opción. Si un tamaño de letra estándar es 16, se puede probar con un tamaño mínimo (9) y un máximo (72) para ver qué ocurre. Posteriormente podemos hacer pruebas con dos tamaños intermedios (32 y 48), para ver qué ocurriría con nuestra página si los usuarios necesitan un tamaño de letra muy superior al estándar.

   e) Cambiar la resolución y el tamaño de la ventana (verificando que no es necesario el *scroll* horizontal). Esto puede hacerse fácilmente con el *plugin* "*Web Developer Extension*" de *Firefox*. Este *plugin*, una vez instalado tiene una opción "*Resize*". Mediante su sub-opción "*Edit Resize Dimensions*" podemos introducir nuevas resoluciones que podemos usar para comprobar cómo se comporta nuestra aplicación ante ellas. Resoluciones interesantes a probar son: 800x600, 1024x768, 1152x864, 1280x720, 1280x768, 1280x960, 1280x1024 y 1600x1200.

Las resoluciones a probar están condicionadas por la máxima resolución de nuestro monitor, por lo que se recomienda poner el mismo a la máxima resolución posible antes de hacer estas pruebas.

f) Relacionado con lo anterior, también se recomienda, si es posible, comprobar cómo se ve nuestra aplicación en múltiples tamaños de pantalla (15 pulgadas, 17 pulgadas,...). Otra posible prueba es redimensionar la página múltiples veces (probando a hacerla cada vez más grande y cada vez más pequeña) para ver cómo se comporta su contenido ante esta situación. Si el aspecto de la página se estropea por ello, entonces habremos encontrado un problema.

g) Probar a visualizar la página sin sus hojas de estilo *CSS* para asegurarse de que aún es legible y usable. Se puede hacer fácilmente mediante el menú "*CSS*" de la "*Web Developer Extension*" de *Firefox* de la que se habló anteriormente.

h) Probar a visualizar la página usando una escala de grises. Para esto podemos usar la herramienta web *GrayBit* (`http://graybit.com/main.php`). En lo referente a colores, también debemos usar herramientas de verificación de color:

1) ***Color Contrast Analyzer*** (`www.visionaustralia.org.au/info.aspx?page=628`): permite verificar que los colores de fondo y texto de una imagen o página web contrastan lo suficiente para ser distinguibles por cualquier persona, según los algoritmos desarrollados por el W3C.
2) ***Vischeck*** (`www.vischeck.com/`): muestra como ven una página web personas con distintos tipos de daltonismo: deuteranopia, protanopia y tritanopia.

i) Usar el teclado para navegar a través de los enlaces y controles de formularios, usando *Tab* para desplazarse.

j) Desactivar *scripts*, *applets*, *Flash*, etc. y comprobar que la página sigue siendo navegable. Para esto es especialmente útil el *plugin NoScript* de *Firefox*

3. Examinar las páginas usando uno o varios navegadores especializados: al menos uno de texto y uno de voz (ver revisión preliminar).

4. Leer y evaluar el contenido de las páginas, para buscar texto no claro o incongruente. Sobre todo debe primar que el texto de las páginas sea lo más claro posible.

Paso 4. Elaborar el informe de conclusiones

Finalmente se incluirá un informe de conclusiones que contemplará los siguientes aspectos:

1. Resumen de los principales problemas encontrados y la solución dada a los mismos

2. Resumen de los aspectos positivos que potencian la accesibilidad de la página

3. Recomendación de fututas actividades para mejorar la accesibilidad de la página:

a) Reparación de barreras de accesibilidad encontradas y que no hayan podido ser solucionadas del todo o adecuadamente.

b) Ampliación de aspectos positivos de accesibilidad

c) Monitorización del sitio.

Por último, a modo de resumen se destacan aquí algunos aspectos muy importantes que no deben dejarse nunca de lado cuando se desarrolla una página, para minimizar los problemas cuando se haga una revisión de cualquier clase sobre la misma. No obstante, todo esto se contempla en el *checklist* del *WCAG*:

1. Poner un texto alternativo para las imágenes
2. Poner texto alternativo en los hipervínculos
3. Revisar los formularios una vez creados (navegación, claridad)
4. Marcos: títulos en los marcos y existencia de *tags* <*noframes*>
5. Desactivar *scripts* y *Flash* y comprobar que la página no pierde funcionalidades
6. Navegar sólo con el teclado y comprobar que se puede acceder a todo el sitio.
7. Hacer especial hincapié en la revisión de la página de inicio (más importante).

Checklist del WCAG 1.0

La siguiente tabla es el *checklist* que el *WCAG* proporciona para verificar las pautas de accesibilidad de la aplicación *web*, pero adaptado a *Word* para poder ser editado fácilmente. Cada punto tiene antes un hiperenlace que va directamente a la web del *WCAG* para proporcionar explicaciones adicionales sobre el mismo (que pueden ir bien para arreglar los problemas detectados).

Puntos de verificación Prioridad 1:

Puntos de verificación Prioridad 1			
En general...	**Sí**	**No**	**N/A**
Proporcione un texto equivalente para todo elemento no textual (Por ejemplo, a través de "alt", "longdesc" o en el contenido del elemento). *Esto incluye:* imágenes, representaciones gráficas del texto, mapas de imagen, animaciones (Por ejemplo, *GIFs* animados), "applets" y objetos programados, "ascii art", marcos, scripts, imágenes usadas como viñetas en las listas, espaciadores, botones gráficos, sonidos (ejecutados con o sin interacción del usuario), archivos exclusivamente auditivos, banda sonora del vídeo y vídeos.		**X**	
Asegúrese de que toda la información transmitida a través de los colores también esté disponible sin color, por ejemplo mediante el contexto o por marcadores.	**X**		
Identifique claramente los cambios en el idioma del texto del documento y en cualquier texto equivalente (por ejemplo, leyendas).			**X**
Organice el documento de forma que pueda ser leído sin hoja de estilo. Por ejemplo, cuando un documento HTML es interpretado sin asociarlo a una hoja de estilo, tiene que ser posible leerlo.			
Asegúrese de que los equivalentes de un contenido dinámico son actualizados cuando cambia el contenido dinámico.			
Hasta que las aplicaciones de usuario permitan controlarlo, evite provocar destellos en la pantalla.			
Utilice el lenguaje apropiado más claro y simple para el contenido de un sitio.			

continua en la próxima página

Puntos de verificación Prioridad 1			
Y si utiliza imágenes y mapas de imagen...	**Sí**	**No**	**N/A**
Proporcione vínculos redundantes en formato texto para cada zona activa de un mapa de imagen del servidor.			
Proporcione mapas de imagen controlados por el cliente en lugar de por el servidor, excepto donde las zonas sensibles no puedan ser definidas con una forma geométrica.			
Y si utiliza tablas...	**Sí**	**No**	**N/A**
En las tablas de datos, identifique los encabezamientos de fila y columna.			
Para las tablas de datos que tienen dos o más niveles lógicos de encabezamientos de fila o columna, utilice marcadores para asociar las celdas de encabezamiento y las celdas de datos.			
Y si utiliza marcos ("frames")...	**Sí**	**No**	**N/A**
Titule cada marco para facilitar su identificación y navegación.			
Y si utiliza "applets" y "scripts"...	**Sí**	**No**	**N/A**
Asegure que las páginas sigan siendo utilizables cuando se desconecten o no se soporten los scripts, *applets* u otros objetos programados. Si esto no es posible, proporcione información equivalente en una página alternativa accesible.			
Y si utiliza multimedia...	**Sí**	**No**	**N/A**
Hasta que las aplicaciones de usuario puedan leer en voz alta automáticamente el texto equivalente de la banda visual, proporcione una descripción auditiva de la información importante de la banda visual de una presentación multimedia.			
Para toda presentación multimedia dependiente del tiempo (por ejemplo, una película o animación) sincronice alternativas equivalentes (por ejemplo, subtítulos o descripciones de la banda visual) con la presentación.			
Y si todo lo demás falla...	**Sí**	**No**	**N/A**
Si, después de los mayores esfuerzos, no puede crear una página accesible, proporcione un vínculo a una página alternativa que use tecnologías *W3C*, sea accesible, tenga información (o funcionalidad) equivalente y sea actualizada tan a menudo como la página (original) inaccesible.			

continua en la próxima página

Puntos de verificación Prioridad 1

Puntos de verificación Prioridad 2:

Puntos de verificación Prioridad 2			
En general...	**Sí**	**No**	**N/A**
Asegúrese de que las combinaciones de los colores de fondo y primer plano tengan el suficiente contraste para que sean percibidas por personas con deficiencias de percepción de color o en pantallas en blanco y negro (Prioridad 2 para las imágenes. Prioridad 3 para los textos).			
Cuando exista un marcador apropiado, use marcadores en vez de imágenes para transmitir la información.			
Cree documentos que estén validados por las gramáticas formales publicadas.			
Utilice hojas de estilo para controlar la maquetación y la presentación.			
Utilice unidades relativas en lugar de absolutas al especificar los valores en los atributos de los marcadores de lenguaje y en los valores de las propiedades de las hojas de estilo.			
Utilice elementos de encabezado para transmitir la estructura lógica y utilícelos de acuerdo con la especificación.			
Marque correctamente las listas y los ítems de las listas.			
Marque las citas. No utilice el marcador de citas para efectos de formato tales como sangrías.			
Asegúrese de que los contenidos dinámicos son accesibles o proporcione una página o presentación alternativa.			
Hasta que las aplicaciones de usuario permitan controlarlo, evite el parpadeo del contenido (por ejemplo, cambio de presentación en periodos regulares, así como el encendido y apagado).			
Hasta que las aplicaciones de usuario proporcionen la posibilidad de detener las actualizaciones, no cree páginas que se actualicen automáticamente de forma periódica.			
Hasta que las aplicaciones de usuario proporcionen la posibilidad de detener el redireccionamiento automático, no utilice marcadores para redirigir las páginas automáticamente. En su lugar, configure el servidor para que ejecute esta posibilidad.			
Hasta que las aplicaciones de usuario permitan desconectar la apertura de nuevas ventanas, no provoque apariciones repentinas de nuevas ventanas y no cambie la ventana actual sin informar al usuario.			
Utilice tecnologías *W3C* cuando estén disponibles y sean apropiadas para la tarea y use las últimas versiones que sean soportadas.			
Evite características desaconsejadas por las tecnologías *W3C*.			
Divida los bloques largos de información en grupos más manejables cuando sea natural y apropiado.			

continua en la próxima página

Puntos de verificación Prioridad 2			
Identifique claramente el objetivo de cada vínculo.			
Proporcione metadatos para añadir información semántica a las páginas y sitios.			
Proporcione información sobre la maquetación general de un sitio (por ejemplo, mapa del sitio o tabla de contenidos).			
Utilice los mecanismos de navegación de forma coherente.			
Y si utiliza tablas...	**Sí**	**No**	**N/A**
No utilice tablas para maquetar, a menos que la tabla tenga sentido cuando se alinee. Por otro lado, si la tabla no tiene sentido, proporcione una alternativa equivalente (la cual debe ser una versión alineada).			
Si se utiliza una tabla para maquetar, no utilice marcadores estructurales para realizar un efecto visual de formato.			
Y si utiliza marcos ("frames")...	**Sí**	**No**	**N/A**
Describa el propósito de los marcos y cómo éstos se relacionan entre sí, si no resulta obvio solamente con el título del marco.			
Y si utiliza formularios...	**Sí**	**No**	**N/A**
Hasta que las aplicaciones de usuario soporten explícitamente la asociación entre control de formulario y etiqueta, para todos los controles de formularios con etiquetas asociadas implícitamente, asegúrese de que la etiqueta está colocada adecuadamente.			
Asocie explícitamente las etiquetas con sus controles.			
Y si utiliza "applets" y "scripts"...	**Sí**	**No**	**N/A**
Para los *scripts* y *applets*, asegúrese de que los manejadores de eventos sean independientes del dispositivo de entrada.			
Hasta que las aplicaciones de usuario permitan congelar el movimiento de los contenidos, evite los movimientos en las páginas.			
Haga los elementos de programación, tales como *scripts* y *applets*, directamente accesibles o compatibles con las ayudas técnicas [Prioridad 1 si la funcionalidad es importante y no se presenta en otro lugar; de otra manera, Prioridad 2].			
Asegúrese de que cualquier elemento que tiene su propia interfaz pueda manejarse de forma independiente del dispositivo.			
Para los "scripts", especifique manejadores de evento lógicos mejor que manejadores de eventos dependientes de dispositivos.			

Puntos de verificación Prioridad 3:

Puntos de verificación Prioridad 3			
En general...	**Sí**	**No**	**N/A**
Especifique la expansión de cada abreviatura o acrónimo cuando aparezcan por primera vez en el documento.			
Identifique el idioma principal de un documento.			
Cree un orden lógico para navegar con el tabulador a través de vínculos, controles de formulario y objetos.			
Proporcione atajos de teclado para los vínculos más importantes (incluidos los de los mapas de imagen de cliente), los controles de formulario y los grupos de controles de formulario.			
Hasta que las aplicaciones de usuario (incluidas las ayudas técnicas) interpreten claramente los vínculos contiguos, incluya caracteres imprimibles (rodeados de espacios), que no sirvan como vínculo, entre los vínculos contiguos.			
Proporcione la información de modo que los usuarios puedan recibir los documentos según sus preferencias (por ejemplo, idioma, tipo de contenido, etc.).			
Proporcione barras de navegación para destacar y dar acceso al mecanismo de navegación.			
Agrupe los vínculos relacionados, identifique el grupo (para las aplicaciones de usuario) y, hasta que las aplicaciones de usuario lo hagan, proporcione una manera de evitar el grupo.			
Si proporciona funciones de búsqueda, permita diferentes tipos de búsquedas para diversos niveles de habilidad y preferencias.			
Localice la información destacada al principio de los encabezamientos, párrafos, listas, etc.			
Proporcione información sobre las colecciones de documentos (por ejemplo, los documentos que comprendan múltiples páginas).			
Proporcione un medio para saltar sobre un *ASCII art* de varias líneas.			
Complemente el texto con presentaciones gráficas o auditivas cuando ello facilite la comprensión de la página.			
Cree un estilo de presentación que sea coherente para todas las páginas.			
Y si utiliza imágenes o mapas de imagen...	**Sí**	**No**	**N/A**
Hasta que las aplicaciones de usuario interpreten el texto equivalente para los vínculos de los mapas de imagen de cliente, proporcione vínculos de texto redundantes para cada zona activa del mapa de imagen de cliente.			

continua en la próxima página

Puntos de verificación Prioridad 3			
Y si utiliza tablas...	**Sí**	**No**	**N/A**
Proporcione resúmenes de las tablas.			
Proporcione abreviaturas para las etiquetas de encabezamiento.			
Hasta que las aplicaciones de usuario (incluidas las ayudas técnicas) interpreten correctamente los textos contiguos, proporcione un texto lineal alternativo (en la página actual o en alguna otra) para *todas* las tablas que maquetan texto en paralelo, en columnas de palabras.			
Y si utiliza formularios...	**Sí**	**No**	**N/A**
Hasta que las aplicaciones de usuario manejen correctamente los controles vacíos, incluya caracteres por defecto en los cuadros de edición y áreas de texto.			

Accesibilidad con Dispositivos Móviles

En caso de que la página a analizar esté destinada a un dispositivo móvil (o tenga una parte o una versión destinada para los mismos), se proporcionan aquí enlaces a herramientas e información útil en estos casos:

1. ***W3C mobileOK Checker***: `http://validator.w3.org/mobile/`
2. ***TAW Ok Basic***: `http://validadores.tawdis.net/mobileok/es`
3. ***Ready.mobi***: `http://ready.mobi/launch.jsp?locale=en_EN`

Otro aspecto a tener en cuenta con estos dispositivos es que si tenemos que evaluar la página en varios de ellos puede ser muy difícil conseguir el hardware necesario. No obstante, para esto nos pueden servir emuladores de los mismos, que permitan comprobar sobre un mismo PC y de forma rápida y sencilla como se ve nuestro sitio en distintos dispositivos móviles con gran fiabilidad. Podemos pues usarlos para enriquecer nuestras pruebas de usabilidad, mostrando cómo se ve nuestra aplicación en un dispositivo móvil. Ejemplos son:

1. **.mobi**: `mtld.mobi/emulator.php`
2. ***The Openwave Phone Simulator***: `developer.openwave.com/dvl/tools_and_sdk/phone_simulator/`

Una vez pasadas las pruebas indicadas en estos enlaces (para las que también pueden tomarse ideas de las pruebas de accesibilidad anteriores), se debe hacer un informe similar a los de la sección anterior indicando:

1. Tipos de problemas encontrados, como se han resuelto y aspectos positivos de la página.
2. Indicar como se detectó cada problema.

8.4. Pruebas de Rendimiento

Explicamos el desarrollo, resultados y cambios derivados de la ejecución de todas las pruebas de rendimiento que hayamos especificado para el sistema, según lo que hemos diseñado en el anteriormente.

Capítulo 9

Manuales del Sistema

9.1. Manual de Instalación

Elaborar un manual que contemple todos los pasos necesarios para instalar nuestro sistema, incluyendo la instalación de otras herramientas o software cualquiera (sea o no comercial) necesario para que funcione. Debemos explicarlo todo paso a paso de forma clara y acompañarlo por capturas de pantalla adecuadas.

9.2. Manual de Ejecución

Este manual contemplará todos los pasos necesarios para el arranque de nuestro sistema, lo que es especialmente importante en caso de sistemas con clientes y servidores o distintos procesos que deban arrancarse independientemente.

Por otra parte, también debemos incluir procedimientos para parar adecuadamente la aplicación.

9.3. Manual de Usuario

El manual de usuario es algo muy importante debido a que es el documento que servirá a los usuarios de nuestro sistema para saber cómo funciona cada una de las partes de nuestra aplicación. Debemos pues describir cómo funcionan todas las opciones de las misma, que parámetros tiene, que cosas debemos hacer para que todas las operaciones funcionen correctamente y cualquier otro aspecto que consideremos oportuno para explicar el funcionamiento del sistema.

No debemos escatimar detalles en este manual ya que es la herramienta para que los usuarios comprendan nuestro sistema. También debemos hacer el mayor uso posible de capturas de pantalla para mejorar nuestras explicaciones.

9.4. Manual del Programador

En este manual debemos describir cualquier aspecto que pueda ayudar a otros programadores a ampliar, modificar o entender aspectos de la construcción de nuestra aplicación. Debemos por tanto hacer una descripción general de los distintos aspectos involucrados en la construcción del sistema que puedan ser más difíciles de entender y también describir los procedimientos necesarios para hacer ciertas ampliaciones que hayamos contemplado en el diseño del sistema (añadir nuevas entidades, nuevos atributos a entidades existentes, nuevos servicios que usen a los ya desarrollados, modificaciones en la interfaz, etc.).

Capítulo 10

Conclusiones y Ampliaciones

10.1. Conclusiones

Conclusiones del sistema: Qué hemos elaborado, si los resultados están dentro de lo esperado, si hemos cumplido las expectativas, justificación de haber escogido las mejores opciones para cada uno de los aspectos del sistema, etc.

10.2. Ampliaciones

Cualquier labor de ampliación que tengamos contemplada en el sistema debe ser descrita aquí, mencionando en qué consiste, cómo ampliará el sistema, qué ventajas nos aporta y porqué no se ha incluido en el sistema diseñado, entre otros aspectos.

Capítulo 11

Presupuesto

A continuación se presentan dos posibles alternativas para el desarrollo del presupuesto del proyecto. La primera de ellas (recomendada) contempla muchos más aspectos y es más completa en líneas generales que la segunda. Seleccionar una u otra depende del criterio del director y del tipo de proyecto. Se presentan ambas para que se seleccione la que se considere más adecuada en cada caso. Otra posible opción es tomar elementos de ambas opciones a conveniencia. **En cualquier caso, consultar al director de proyecto acerca de cuál de las dos es más adecuada para el proyecto desarrollado.**

11.1. Desarrollo de Presupuesto Detallado (Opción 1) (Recomendado)

A la hora de elaborar un presupuesto hay que tener en cuenta dos aspectos importantes:

1. ¿Cuánto cuesta desarrollar el proyecto? **(Presupuesto de Costes)**
2. ¿Cuánto voy a cobrar por los trabajos? **(Presupuesto del Cliente)**

El alumno debe realizar ambos. Por lo que respecta al presupuesto del cliente, se hará constar en esta sección y deberá estar basado en el presupuesto de costes realizado. El presupuesto de costes puede realizarse en esta sección o figurar en un anexo específico, a criterio del director del proyecto.

Dicho presupuesto debe ser claro y exhaustivo, adelantándose a las dudas del cliente y presentando todos los elementos, de manera que no exista ningún "punto oscuro". Para ello deben constar únicamente ítems que entienda el cliente. Como ejemplo, el coste de amortización de los equipos de desarrollo no es un dato que deba constar en este presupuesto, sino en el de costes. Un curso de formación, el análisis del sistema, la instalación de un SGBD, etc. si son elementos que entregamos al cliente, sí deben constar en este apartado.

En general el presupuesto del cliente debe cubrir el presupuesto de costes de desarrollo para que el proyecto resulte rentable y debe tener un margen de beneficio con respecto a aquel.

El presupuesto del cliente no debe ser una tabla de valores monetarios vacía, sino que debe ser acompañado por una explicación de estos valores.

Las características fundamentales que debe cumplir son:

1. **Completo:** Desde la presentación de nuestra empresa, hasta la solución que pensamos darle al proyecto del cliente, ejemplos de proyectos ya realizados, presupuesto cerrado indicando qué incluye y tan importante como esto, qué no incluye.
2. **Claro:** Siempre hay que pensar que el cliente no sabe. Si sabe puede pensar que somos muy básicos, pero si no sabe y le pasamos un presupuesto excesivamente

técnico, no le podremos hacer llegar nuestra idea de manera que él pueda visualizar el resultado.

3. **Conciso:** No hay que andarse por las ramas. A nosotros nos interesa poder transmitir la mayor cantidad de ideas en el menor espacio posible. Para ello, es muy recomendable utilizar tablas y/o puntos.

A la hora de desarrollar el presupuesto, debemos pensar en hacer referencias a:

1. **Metodología de trabajo:** Explicar al cliente como vamos a hacer las cosas, comentar por ejemplo que primero se va a diseñar la web o la base de datos, que hasta que no se apruebe (firmado), no se va a pasar a la siguiente fase.

2. **Tiempos de ejecución:** Hay que cumplirlos. También hay que ajustar mucho e indicar cuáles son nuestras responsabilidades y cuáles son las suyas en cuanto a entregas de material, validaciones etc., y la influencia que tendrá en los tiempos, el retraso en las entregas y las aprobaciones.

3. **Presupuesto detallado propiamente dicho:** Es muy útil tabular cada una de las secciones que van a componer el proyecto y desglosarlo por su coste.

4. **Formas de pago (Si procede):** Indicando también los plazos que habrá que cumplir.

Todos estos apartados, pueden no llevar este orden, incluso estar mezclados, pero es importante tener transmitir todo esto claramente al cliente. Un ejemplo de un desglose de un presupuesto en una tabla se muestra a continuación:

El presupuesto debe contener:

1. Cuando proceda, un cuadro de precios de las unidades de medida correspondientes: componentes de hardware, elementos de software, horas persona de diferentes categorías, elementos auxiliares y otros.

2. Cuando proceda, costes de unidades lógicas con entidad propia dentro del proyecto, con la descomposición correspondiente de componentes de hardware, elementos de software, horas persona, elementos auxiliares y otros.

3. El presupuesto propiamente dicho debe contener la valoración económica global, descompuesta siguiendo la estructura de desglose de los elementos utilizada en la planificación y ejecución del proyecto.

4. El presupuesto debe especificar claramente las bases con las que se confecciona el mismo.

Item	**SubItem**	**Concepto**	**Cantidad**	**P. Unidad**	**TOTAL**
01		***Desarrollo Aplicación***			7400,00 €
	001	*Análisis*	20,00 €	60,00 €	1200,00 €
	002	*Diseño*	25,00 €	60,00 €	1500,00 €
	003	*Implementación*	100,00 €	37,00 €	3700,00 €
	004	*Pruebas*	25,00 €	40,00 €	1000,00 €
02		***Formación***			8000,00 €
	001	*Directivos*	20,00 €	50,00 €	1000,00 €
	002	*Usuarios*	40,00 €	50,00 €	2000,00 €
	003	*Mantenimiento*	100,00 €	50,00 €	5000,00 €
					continua en la próxima página

Item	**SubItem**	**Concepto**	**Cantidad**	**P. Unidad**	**TOTAL**
				Subtotal	30.800,00 €
				IVA(16 %)	4.928,00 €
				TOTAL	35.728,00 €

Cuadro 11.1: Tabla de ejemplo de resumen de presupuestos

11.2. Desarrollo de Presupuesto Simplificado (Opción 2)

En esta sección debemos rellenar esta tabla Excel para calcular el coste económico de desarrollar el sistema planteado en la documentación. Debemos considerar cosas como las horas de análisis, diseño y programación, el personal necesario, materiales, programas, equipos, etc.

Si fuese necesario, podemos incluir en esta sección una planificación temporal de las distintas etapas del proyecto (análisis, diseño, etc.).

Concepto	**Cantidad**	**P. Unidad**	**TOTAL**
Horas de Programador		0,00 €	0,00 €
Horas de Análisis y Diseño		0,00 €	0,00 €
...			0,00 €
			0,00 €
			0,00 €
		Subtotal	0,00 €
		IVA(16 %)	0,00 €
		TOTAL	0,00 €

Cuadro 11.2: Tabla de ejemplo de resumen de presupuestos

Capítulo 12

Apéndices

12.1. Glosario y Diccionario de Datos

Por orden alfabético, todos los términos que se consideren importantes en la aplicación con una descripción breve de su significado dentro de la aplicación.

1. **Término1**: Descripción del significado.
2. **Término2**: Descripción del significado.

12.2. Contenido Entregado en el CD-ROM

12.2.1. Contenidos

Descripción del contenido de los diskettes o *CDs* (directorios y para qué sirve cada cosa), descripción de esta documentación y de cualquier material que adicionalmente se entregue en la presentación. En esta sección se presenta una estructura de directorios de ejemplo para el CD que se puede seguir para distribuir todos sus contenidos en el mismo. Conviene por tanto tenerla en cuenta desde el principio de la implementación.

Introducción

La estructura de directorios del proyecto debe poder recoger todos los ficheros relacionados con el proyecto, clasificándolos por su propósito dentro del mismo. Los tipos más frecuentes son: ficheros fuente, ficheros de configuración, ficheros de documentación...

Se deben crear directorios para contener cada uno de los tipos de ficheros. Tener una estructura estandarizada de los directorios del proyecto es importante por varias razones:

1. Ayuda a localizar la información del proyecto. Por ejemplo, los ficheros fuente siempre deben estar en la carpeta *src*.
2. Ayuda a los desarrolladores a determinar donde debe ir cada fichero.
3. Permite crear scripts de construcción estandarizados.

Recomendación estructura general directorios del CD

NOTA: El nombre del CD debe corresponder con el del proyecto.

Directorio	**Contenido**
./ Directorio raíz del CD	Contiene un fichero leeme.txt explicando toda esta estructura. Se puede incluir *autorun* e icono del proyecto si existe.
./<nombre_proyecto>	Contiene toda la estructura de directorios del proyecto para desarrollo. **Ver la tabla Recomendación de estructura de directorios de desarrollo**. <nombre_proyecto> debe sustituirse por el nombre corto del proyecto.
./instalacion	Ficheros utilizados para la instalación del proyecto.
./documentacion	Contiene toda la documentación asociada al proyecto. Es necesario incluir un fichero con el documento final del proyecto (en formato *.doc* o *.docx* de *Word* o bien *.sxw* de *Open Office*, por ejemplo) además de un fichero *.PDF*
./documentacion/img	Directorio que contiene las imágenes utilizadas en la documentación. Estas imágenes tendrán formato *.png* si son capturas de pantalla, *.wmf* si son diagramas o esquemas y *.jpg* sólo si son fotografías.
./documentacion/uml	Ficheros que genera la herramienta (Rose, *ArgoUML*, etc.) con la que se han generado los diagramas UML y de entidad relación.

continua en la próxima página

Directorio	Contenido
./presentacion	Directorio que contiene la presentación en *Powerpoint* o equivalente utilizada el día de la defensa del proyecto, si está disponible en el momento de realizar el CD.
./herramientas	Contiene los ficheros de instalación de las herramientas utilizadas para el desarrollo o puesta en marcha del proyecto (lógicamente sólo las que sean distribuibles).
./herram/desarrollo	Ficheros de instalación de las herramientas utilizadas en el desarrollo
./herram/explotacion	BD, servidor Web y herramientas en general.

Recomendación de la Estructura de Directorios de "desarrollo"

Se muestra aquí el contenido del directorio de desarrollo de la tabla anterior, incluyendo todos los directorios que deben depender del mismo. Algunos de los elementos sea han incluido suponiendo que se están usando ciertas tecnologías Java. En caso de no usarlas, buscaremos un equivalente existente (si lo hay) en la que estemos usando nosotros.

Directorio	Contenido
./ Directorio raíz de "desarrollo"	Contiene los ficheros de proyecto del IDE utilizado.
./build	Contiene el build.xml de *ant* (si lo usamos). Debemos situarnos dentro para poder invocarlo.
./conf	Contiene los diferentes ficheros de configuración del proyecto. Podría contener distintos subdirectorios, en función de la tecnología usada. En este ejemplo se muestra un ejemplo de un proyecto *Web* hecho con tecnologías *Java*: **web**: contiene los ficheros de configuración de la aplicación Web (por ejemplo: web.xml). **ear**: contiene los ficheros de configuración de una aplicación empresarial (por ejemplo: application.xml). **ejb**: contiene los descriptores de despliegue de los EJB.
./dist	Directorio donde se sitúan los ficheros para la distribución del proyecto. Por ejemplo: los ficheros *.war* o *.ear*.
./doc	Contiene toda la documentación relativa al proyecto, incluyendo los ficheros generados por herramientas de generación de documentación automática como *Javadoc* o similar.
./lib	Bibliotecas externas (*.jar*, *.dll*, ...) necesarias para compilar y distribuir, de las que depende este proyecto.
./compile-lib	Bibliotecas externas (*.jar*, *.dll*, ...) necesarias para compilar pero que no deseamos distribuir.
./src	Ficheros fuente
./src/java	Todos los ficheros *Java*, lógicamente agrupados en los paquetes correspondientes.

continua en la próxima página

Directorio	**Contenido**
./src/sql	Este directorio contiene los scripts de *SQL* que permiten construir y meter los datos iniciales en la base de datos del proyecto (si existe).
./web	Este directorio contiene los ficheros (*.JSP*, *.ASPX*, *.HTML*, ...) de la *Web* (si el proyecto incluyese una).
./web/images	Contiene las imágenes utilizadas por los ficheros de la web del proyecto.
./classes	Directorio donde se guardan los ficheros compilados (como por ejemplo los *.class*)
./test	Directorio base para todos los ficheros utilizados en la automatización del proceso de prueba.
./test/java	Contiene todas las pruebas unitarias utilizadas en el proceso de prueba automatizado.
./test/sql	Scripts *SQL* utilizados en la carga de datos de prueba
./bak	Directorio donde se pueden guardar versiones antiguas de los ficheros fuente del proyecto.

12.2.2. Código Ejecutable e Instalación

Descripción de los contenidos del código ejecutable y de la instalación de la aplicación en un ordenador. Breve manual de instalación y puesta en marcha de la aplicación (solamente unos pasos sencillos que faciliten este proceso, sin explicar nada (para eso está el manual de instalación propiamente dicho).

12.2.3. Ficheros de Configuración

Descripción de todos los ficheros necesarios para poder hacer funcionar la aplicación (ficheros de configuración, ficheros de datos, etc.).

Este índice alfabético está generado automáticamente por el procesador de textos e incluirá todos los términos que nosotros marquemos adecuadamente con la orden key (conteniendo entre llaves la palabra a indexar). No debemos pues incluir palabras "a mano" en él. El documento tiene una serie de ejemplos para rellenar un índice de ejemplo y ver como funciona. Podemos encontrar más información sobre los índices en LaTeXen `http://en.wikibooks.org/wiki/LaTeX/Indexing`. No debemos olvidarnos de ejecutar el comando makeindex para que éste salga correctamente.

NOTA: Quitar esta explicación en la documentación final.

Bibliografía

[1] Apellido, Nombre; Apellido2. Nombre2; .... Titulo del Libro o Articulo. Editorial y/o lugar de Publicacion, Año de Publicación. ISBN (si se tiene, se pone al final de la entrada correspondiente).

[2] Lamport, Leslie. *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2ª Edicion, 1994.

[3] Redondo, J. Manuel; De Tal y Cual, Menganito. *Ejemplo para la plantilla de PFC en LaTeX*. Universidad de Oviedo. 2009.

[4] Apellido, Nombre; Apellidos1, Nombre1; .... *Título de la página Web*. URL, Año en el que se consultó o se público el articulo que se referencia (4 cifras)

[5] Redondo, J. Manuel; De Tal y Cual, Menganito. *Título de la página Web de ejemplo*. `www.unaurlcualquiera.com`, 2007.

[6] Hassan, Y. *Guía de Evaluación Heurística de Sitios Web*. `www.nosolousabilidad.com/articulos/heuristica.htm`, 2008

LaTeXpermite un manejo avanzado de las referencias bibliograficas, lo que nos permite hacer esta sección de una forma muy distinta a Word. Incluiremos en una misma sección libros y artículos usados de alguna forma durante el desarrollo del proyecto o su documentación y también páginas Web consultadas para cualquier aspecto relacionado con el desarrollo del sistema o su documentación. Aqui se incluyen algunos ejemplos tanto de como se construyen las entradas y de como se citan, pero para una referencia mucho más completa de posibilidades se recomienda consultar: `http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management`. En lo relativo páginas web, si tenemos más datos que permitan localizar la información dentro de la página que se cita en la referencia correspondiente, podemos ponerla donde consideremos oportuno dentro de la misma. Para ilustrar como se citan las referencias en el texto, se recomienda buscar la referencia Hassan08 (esta referencia es real (se usa dentro del documento) y debe dejarse aquí siempre que usemos el cuestionario que la menciona en la sección de usabilidad) o lamport94, que son reales dentro de este documento y ver como se ha hecho.

Capítulo 13

Código Fuente

El código fuente tiene que ir dividido por paquetes y por archivos, con un formato que haga que resulte legible, tal y como se muestra a continuación en este ejemplo. En la mayoría de las ocasiones, es posible que no sea conveniente colocar la totalidad del código fuente en esta sección, sino solo una parte, que contenga aquellas clases más significativas e importantes. En cualquier caso, conviene consultar al director del proyecto este aspecto. Para dar un formato al código apropiado, podemos probar a copiarlo directamente de nuestro editor y ver si el texto acarrea sus propiedades de estilo (no funcionará en todos los editores), o bien usar programas como *Scintilla* (*Windows*) `http://www.scintilla.org/` o el editor *Kate* (para *KDE*, *Linux*) `http://kate-editor.org/`.

13.1. Paquete Ejemplo 1:

13.1.1. Fichero "A.cs":

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ControlesAvisos
{
     public class Usuario: ICloneable
     {
         public Usuario()
         {
             this.nombre = "";
             this.clave = "";
         }

         public Usuario(string nombre, string clave)
         {
             this.nombre = nombre;
             this.clave = clave;
         }

         private string nombre;
         public string Nombre
         {
             get
             {
                 return nombre;
             }
```

```
            set
            {
                nombre = value;
            }
        }

        private string clave;

        public string Clave
        {
            get
            {
                return clave;
            }

            set
            {
                clave = value;
            }
        }

        public Object Clone()
        {
            Usuario u = new Usuario(this.Nombre, this.Clave);

            return u;
        }
    }
}
```