Aplicación Web para la ejecución y gestión de auditorías de calidad internas

Autor: Juan Carrey Labarta

Director: D. Ignacio Vilalta

Ponente: José Javier Merseguer Hernáiz

Empresa: Comex Grupo Ibérica

Centro Politécnico Superior

Universidad de Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Ingeniería Informática
Curso 2009-2010, Mayo 2010

GESTIÓN DE AUDITORÍAS DE CALIDAD INTERNAS RESUMEN

Se trata del desarrollo e implantación de una aplicación Web para la gestión de auditorías de calidad internas que actualmente la empresa Comex Grupo Ibérica realiza de forma manual.

Esta aplicación permitirá la administración de la misma de forma privada, permitiendo gestionar al personal (auditados/auditores), los proyectos y procesos (partes del proyecto) que se auditan, los tipos de auditoría existentes así como las preguntas que se realizan en cada auditoría.

Además se podrán gestionar diversos parámetros para la realización de avisos automáticos que sirven para recordar a los interesados de hitos o fases de las auditorías.

Por otra parte y de forma separada existirá una parte pública donde se podrán ejecutar auditorías, comprobar el seguimiento de las no conformidades (respuestas negativas a las preguntas) del proyecto y acceder a diversos informes y métricas calculadas por el sistema.

La aplicación obtendrá información de los proyectos y personal de la empresa a través de un servicio Web existente, almacenándose en cada caso la información relevante.

La arquitectura del sistema se basa en un diseño de tres capas:
Una capa de presentación en HTML / JSP, utilizando Ajax y ejecutándose sobre un servidor Tomcat.

Una capa de control, basada en componentes de Spring, donde se intercambiarán los datos con la capa de presentación y se ejecutarán las acciones correspondientes para obtener los datos.

Una capa de acceso a datos basada en el patrón de diseño Data access object (DAO) y accediendo a ella a través de una fachada (patrón facade).

Además existen dos capas de seguridad intermedias para el acceso a la parte privada, una previa a la capa de vista y otra en la capa de acceso a datos.

El motor de base de datos utilizado es SQL Server 2005, para acceder a él utilizaremos Java Persistence Api (JPA) como mapeador objeto-relacional (Object Relational Mapping ORM) utilizando la implementación de Hibernate, aunque realizando una implementación fiel al estándar Java Persistence API (JPA) sin utilizar aspectos propios a la implementación de Hibernate.

Se han realizado las pruebas necesarias para la validación y verificación del buen funcionamiento del software.

Esta aplicación pretende solventar la problemática existente de la distribución de las hojas Excel del método manual que dificultan en gran medida la realización de seguimientos, informes generales, cálculos de métricas, fechas de auditorías, etc.

Tabla de contenidos

Introducción

Objetivo y alcance del proyecto

El objetivo de este proyecto consiste en poner en marcha una aplicación Web de gestión que asista al personal de la empresa Comex Grupo Ibérica a la configuración y realización de auditorías de calidad internas, mejorando el sistema actual y asegurando el mantenimiento de la funcionalidad existente.

Los objetivos específicos de este proyecto son:

Resolver los problemas existentes en el sistema actual.
Facilitar la gestión de auditorías de calidad mediante un sistema de información
Mejorar el sistema actual con diversas herramientas

El marco que define al proyecto se basa en el desarrollo y puesta en marcha del sistema de gestión, asegurando el buen funcionamiento del mismo, obteniendo como resultado una aplicación que, cumpliendo los objetivos marcados, pueda ser en la medida de lo posible, sostenible y ampliable en un futuro.

Ámbito y motivación

El proyecto se enmarca en la empresa **Comex Grupo Ibérica,** donde actualmente el proceso de gestión de auditorías se realiza de forma manual.

A continuación se explican los diversos problemas que surgen del método de gestión actual de las auditorías en la empresa.

La información se recoge en diversas hojas Excel que se almacenan junto a la información de cada proyecto, lo cual dificulta la localización de una auditoría.

Los resultados de las auditorías se reflejan posteriormente en el "Informe General de Auditorías", replicando de está forma la información, lo cual incurre en posibles inconsistencias.

Semestralmente se calculan una serie de métricas, que se realizan también de forma manual, llevando consigo el problema de tener que revisar una a una todas las hojas Excel de auditorías anteriores, situándose cada una en un lugar diferente dependiendo del proyecto, lo cual hace casi imposible realizar una comparación entre proyectos y muy complicado el cálculo de diversas métricas.

Cada proyecto además tiene otra hoja Excel con el seguimiento de las no conformidades, que recogen las respuestas negativas de las auditorías ejecutadas sobre dicho proyecto, por lo que resulta complicado realizar el seguimiento dado que todo está en hojas Excel independientes, con información replicada.

Otro de los problemas existentes con el sistema actual es que el responsable de calidad debe estar atento a las fechas planificadas para las auditorías y ejecutarlas conforme a la planificación.

Esta aplicación pretende solucionar toda esta problemática dado que toda la información resultante de las auditorías se encuentra centralizada en una base de datos única, evitando la replicación, por lo que la explotación de datos será mucho

más sencilla. Además podrá enviar avisos a las personas interesadas a modo de recordatorio para cumplir con las planificaciones establecidas.

Herramientas y tecnologías

Herramientas

Se listan a continuación las diversas herramientas empleadas durante el proyecto, con sus respectivas referencias a su documentación.

Apache[1]

Apache Tomcat se ha utilizado como Servidor Web, se trata de un servidor de aplicaciones Web de uso extendido.

OpenSVN[2] (Open SubVersion) y Tortoise[3]

Debido a que en la empresa en la que se enmarca el proyecto trabajan con un sistema de control de versiones se ha optado por aprender a utilizarlo y usarlo para este proyecto. Se utilizan OpenSVN como servicio gratuito para almacenar las versiones y Tortoise para la gestión local de los ficheros.

Navegadores Web

Para la visualización del resultado y realización de pruebas se han utilizado Internet Explorer[4] y Mozilla Firefox[5].

Microsoft Office 2003 y Acrobat Reader

Para la generación de documentos se han utilizado estas herramientas de edición de texto.

Eclipse[6]

Como entorno de desarrollo se ha utilizado Eclipse debido a las grandes ventajas que nos ofrece para el desarrollo de aplicaciones java, una de las principales ha sido el poder lanzar un servidor tomcat en modo depuración, en el propio entorno de desarrollo, lo cual ha facilitado enormemente la resolución de errores.

Edge Diagramer[7] y GranttProject[8]

Hemos escogido estas dos aplicaciones para la generación de diagramas y calendarios debido a que ya las conocíamos anteriormente y nos parecieron unas herramientas sencillas de manejar, completas y muy útiles.

Testlink[9]

Para la planificación y ejecución de pruebas hemos utilizado Testlink, se trata de una aplicación Web de gestión donde se introducen los planes de prueba con los pasos a seguir en cada caso de prueba, para que cualquier persona con acceso a la aplicación a la que se le hayan asignado la ejecución de diversos casos de prueba pueda ejecutarlos y redactar los resultados.

Los resultados se obtienen en unos informes generados por la aplicación, esta herramienta que se está empezando a usar en la empresa, por lo que hemos decidido emplearla nosotros también.

Microsoft SQL Server Studio

Para la gestión de la base de datos en general hemos usado este software que ya estaba previamente instalado en el sistema y nos pareció bastante bueno para manejar SQL Server.

Tecnologías

Persistencia: Java persistence Api (JPA)[10] Object Relational Mapping

Se ha decidido utilizar JPA como Object Relational Mapping (ORM) debido a su gran potencia y sencillez, nos hemos intentado abstraer lo más posible del proveedor de persistencia (implementación), dado que de esta forma nos es muy cómodo cambiar.

Proveedor de persistencia: Hibernate[11]

Por su madurez y porque otros proveedores nos dieron problemas anteriormente (para la misma implementación), además será la tecnología usada en la empresa en un futuro y es gratuito.

Base de datos: SQL Server 2005

Como gestor de base de datos, la empresa decidió almacenar la información para las auditorías en su base de datos SQL Server 2005 para mantener en ésta todos los datos internos.

Servicios Web: WSS4J[12] y Axix 1.4[13]

Para el acceso a los Servicios Web existentes, hemos utilizado Web-Service Security for Java (WSS4J) por su fácil configuración a través de un fichero de despliegue, mientras que en otros la configuración iba metida en el código, además buscando información sobre como compatibilizar con Web Service Enhacement (WSE) 3.0 era de las pocas tecnologías para las que se dejaba clara la solución a uno de los problemas que hay en la interoperabilidad entre estos servicios Web.

Spring Framework[14]

Como tecnología Web para la creación de sistemas Web complejos que facilitan la estructuración en tres capas Modelo – Vista – Controlador, (MVC[17]) así como las grandes ventajas que ofrece sobre otros Frameworks similares como Struts, el cual se planteó inicialmente como opción debido a que iba a ser lo que se iba a utilizar en la empresa en proyectos futuros, pero se cambió a Spring porque era nuestra primera opción dado que mejoraba sustancialmente parte de los problemas que tiene Struts y además la empresa decidió utilizar Spring en sus futuros proyectos.

Spring Security[15]

Dentro de Spring, existe una herramienta que sirve para configurar permisos y cuentas con el fin de restringir el acceso a diversas vistas de la Web a determinados

usuarios, es por ello que decidimos escoger esta opción como seguridad en la capa de vista, porque nos venía con el propio Framework , ya estaba implementado y además era bastante bueno.

JUnit

Herramienta que nos facilita en gran medida las pruebas unitarias y de integración, realizando para cada caso de prueba una clase JUnit que realice las pruebas sobre un módulo en concreto, ya sea de forma unitaria, o de forma integrada (si el modulo está integrado en el sistema).

Otras tecnologías

Java con lenguaje de programación para todo el control y acceso a datos.
Java Server Pages (JSP) para las vistas, que mezclan código HTML con Java
Cascading Style Sheets (CSS) para el diseño
Ajax para añadirle usabilidad al sistema
JavaScript para añadirle usabilidad al sistema (Para hacer llamadas Ajax y mostrar calendarios)
HyperText Markup Language (HTML) como código final generado por el servidor para el navegador Web.

Se han utilizado estos lenguajes de programación, en principio estipulados por la empresa para la realización del proyecto dado que es con lo que habitualmente se trabaja en aplicaciones Web.

Estructura del documento

La estructura general del documento consta de diversas secciones donde cada una está subdividida en subapartados, este documento se divide en una introducción donde se explica brevemente el contexto del proyecto, seguida de varias secciones que se relacionan con diversas fases del proyecto, que se describen a continuación.

En la sección de análisis se describe el proceso en el cual el sistema manual actual se transforma en una serie de requisitos que deberá cumplimentar la aplicación a desarrollar.

En el diseño de la base de datos, se describen brevemente los pasos seguidos durante el diseño del modelo de datos destacando los aspectos más relevantes.

En la fase del diseño del sistema, se explican los métodos que hemos llevado a cabo para obtener un esquema general y la forma de construcción del sistema sin llegar al detalle de la implementación, detallando los casos más particulares y que nos han parecido de mayor interés.

En el apartado donde se describe la implementación, hemos considerado explicar los casos problemáticos más destacados e interesantes en la implementación, explicando la solución adoptada.

La última de las secciones referente a las pruebas contiene información relevante de las pruebas indicando los resultados generales de su ejecución, sin entrar en el detalle de cada caso específico de prueba.

En el último apartado del documento ponemos de manifiesto nuestras reflexiones y conclusiones sobre el proyecto, así como una valoración y estimación del trabajo realizado.

Los anexos contienen la información completa de las fases del proyecto (análisis, diseño, pruebas, documentación, etc.), con todos los detalles pertinentes.

Planificación

Durante el proceso de planificación del proyecto, planteamos el ciclo de vida del software a desarrollar, los periodos de tiempo durante los cuales se realizarían cada una de las tareas y se escogen las herramientas y tecnologías a utilizar.

El proyecto se planteo para tener una duración aproximada de 400 horas, siguiendo un proceso iterativo incremental donde las primeras fases previas al desarrollo se hicieron siguiendo el modelo en cascada, donde se establecieron los requisitos y diseño iniciales, que poco a poco en las sucesivas iteraciones se van refinando y mejorando para ir desarrollando versiones intermedias, para llegar a obtener el producto final.

Estas iteraciones, donde se refina el análisis y los requisitos, vienen recogidas en la fase de implementación, dado que es imposible planificar todas las pequeñas fases por cada versión intermedia a realizar.

Cada módulo realizado, se prueba de forma unitaria y posteriormente se integra al sistema, realizando las pruebas de integración, consiguiendo en cada integración versiones intermedias.

Fases del proyecto

Planificación

En esta etapa se planifican los calendarios y se establece la gestión de configuraciones.

Análisis

En esta fase se intentan captar todas las necesidades del cliente, generando un análisis de requisitos, se escogen las herramientas a utilizar y se establecen las tecnologías que se utilizan, se trata de un análisis inicial que puede ser modificado en posteriores revisiones, que se realizan durante el desarrollo.

Diseño

En el diseño se generan todos los documentos necesarios para plasmar la solución diseñada, lo más claramente posible, realizando diagramas de clases, esquemas conceptuales, casos de uso, etc.

Se escogen los métodos para realizar la implementación (Arquitectura del sistema, patrones de diseño, etc.), y se definen cada uno de los nombres en el diccionario de datos.

Se genera además el modelo físico de la base de datos donde se almacenará la información.

Implementación

En esta fase se desarrollara todo el código fuente siguiendo las pautas del diseño, apartado por apartado (modulo), reanalizando y rediseñando cada uno de forma particular y en detalle, posteriormente se prueba dicho apartado y se integra al sistema, hasta completar todos los apartados y cumplimentar los requisitos.

Pruebas

Este apartado consiste en encontrar todos los posibles errores del sistema final que hayan podido quedar candentes, por ello se generará un plan de pruebas posterior al análisis o al menos antes de realizar las pruebas.

Así mismo se prueba la usabilidad del sistema de forma que resulte amigable la interacción con el mismo.

Implantación

En este proceso se tiene en cuenta la fase de instalación del aplicativo en el servidor, configuración del servidor, etc.

Documentación

En esta fase se tiene en cuenta la documentación del proyecto, redacción de documentos y revisión de los mismos, la documentación del proyecto viene un poco mezclada con las demás secciones, ya que en muchos de los procesos se generan diversos documentos, en esta fase se revisan y se redactan correctamente todos ellos, a parte de generar la memoria final del proyecto.

Mantenimiento

Esta fase del proyecto no viene contemplada en la planificación, dado que su extensión puede llegar a ser “indefinida”, pero forma parte del proyecto y hay que tenerla en cuenta.

Calendario fijado

El proyecto se enmarca en el desarrollo de un proyecto fin de carrera de unas 400 horas aproximadamente para una única persona, así pues se ha desarrollado un calendario planificando las distintas etapas del proyecto.

Como hemos dicho anteriormente, la etapa de implementación enmarca rediseños en detalle y pruebas unitarias y de integración.

El siguiente diagrama muestra la planificación del proyecto estimada en las fases iniciales del proyecto.

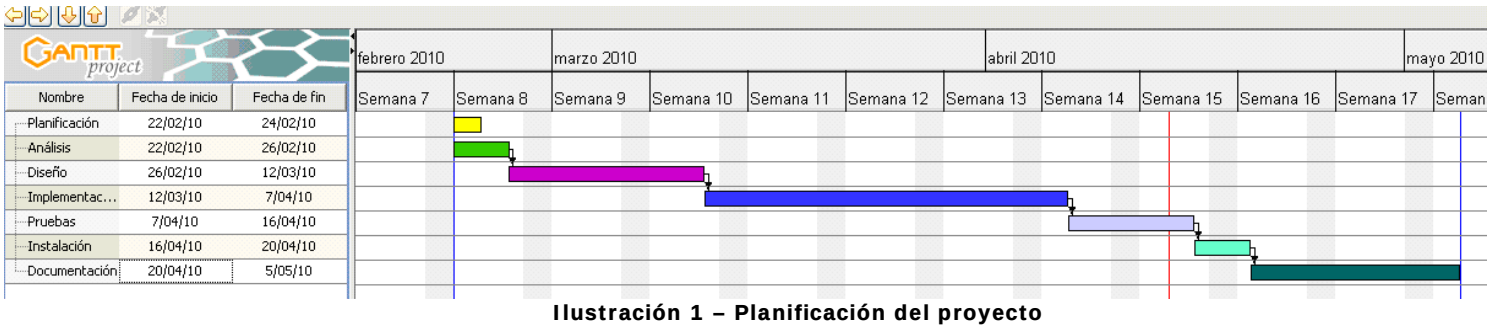


Ilustración 1 – Planificación del proyecto

Hitos del proyecto

Para el correcto seguimiento del proyecto, se han planificado diversos puntos críticos del proyecto donde se marcan una serie de objetivos a cumplimentar en determinadas fechas.

Estos hitos están enfocados a conseguir un cierto grado de completitud y estabilidad en el proyecto.

Los hitos planificados son los siguientes.

Hito 1: **2 Marzo**

Análisis completado, con los requisitos y la planificación realizada, gestión de configuraciones.

Hito 2: **8 Marzo**

Diseño de la Base de datos, esquema conceptual, lógico y físico.

Hito 3: **12 Marzo**

Diseño general del sistema, tecnologías, esquemas del sistema.

Hito 4: **19 Marzo**

Diseño detallado del sistema, casos de uso.

Hito 5: **9 Abril**

Sistema implementado con pruebas unitarias y de integración.

Hito 6: **5 Mayo**

Pruebas, documentación básica y documentación del PFC. Sistema implantado.

Gestión de configuraciones

Dado que la empresa sigue un estilo en la gestión de configuraciones, hemos redactado un documento en el que se explica detalladamente la localización de cada recurso del proyecto.

De esta forma el proyecto resultante queda bien estructurado y organizado según la configuración descrita en el documento.

Análisis

En este apartado se lleva a cabo la definición, análisis y validación de los requisitos a partir de la información facilitada por los usuarios participantes. Se obtiene un catálogo detallado de los requisitos, a partir del cual se pueda comprobar que los productos generados en las actividades de modelización se ajustan a los requisitos de usuario.

Durante esta fase se estableció contacto con el cliente, llevando siempre una agenda y una serie de preguntas.

En las reuniones se establecieron los requerimientos de la aplicación y se detalló el funcionamiento del sistema actual, a continuación se detallan estos requisitos.

Requisitos funcionales parte pública

1.- El sistema tendrá una gestión de avisos por email al personal correspondiente, ya sea auditor o auditado según la siguiente lógica:
1.1 Al dar de alta una nueva auditoría, se enviará un correo a los auditados. El correo contendrá un link a una página que permita modificar la fecha de la auditoría correspondiente.
1.2 Al modificarse las fechas de auditación de una auditoría se enviará un correo a los auditores. El correo contendrá información de las fechas de la auditoría, para recordárselas al auditor.
1.3 Unos días antes (también configurables) de que tenga lugar la fecha de auditación de una auditoría, se enviará un email tanto a los auditores como auditados para la realización de la misma.
1.4 Si no se contesta al aviso del punto 2.1 (pasan un número determinado y configurable de días sin obtener respuesta), se reenviará el aviso para realizar la planificación de nuevo, cuando se hayan enviado un número de avisos, también configurable, se enviará un aviso al auditor para su conocimiento.
1.5 Al terminar una auditoría (completarla) el auditor recibirá un email (o bien desde la aplicación) para acceder al seguimiento de no conformidades de la auditoría ó proyecto.
2.6.- Pasados un número de días configurable desde que se completa el seguimiento el auditor recibirá un email para recordarle de comprobar como se está llevando a cabo dicho seguimiento.

2.- En la parte pública se permitirá la creación, modificación y ejecución de auditorías, realizadas o bien sobre un proyecto (No especiales) o bien que no requieran proyecto (especiales).
Por defecto en la creación de auditorías vendrán marcadas todas las no especiales.
En la ejecución de la auditoría se permitirá la navegación mediante pestañas, dónde cada pestaña corresponderá a un proceso distinto.
Se podrán buscar proyectos y personal a través del servicio Web disponible y mediante el código de recurso.

3.- Desde la parte pública se podrán consultar diversos informes:

- Resumen de resultados de todas las auditorías: contendrá información de cada auditoría indicando el número de no conformidades de cada tipo, el riesgo previo y posterior a la auditoría, etc. es decir la información que aparece en la hoja Excel del resumen de resultados del método manual.

- Resumen de resultados de las auditorías por tipo de proyecto.

- Resumen de resultados de las auditorías por jefe de proyecto.

- Resumen de una auditoría, se trata de la información de las no conformidades asociadas a una auditoría, con la descripción el tipo y la causa de la no conformidad, así mismo se podrá observar el estado del proyecto actual y el estado de esta auditoría según el número de no conformidades de cada tipo.
  El estado/Riesgo de un proyecto/auditoría se calcula mediante una formula que relaciona el número de no conformidades de un determinado tipo con el estado a través de un número que será el máximo número de no conformidades de un tipo para el cual se puede estar en ese estado. El estado resultante será el PEOR de todos los estados que se cumplan, siendo el peor estado aquel que requiera MENOR número de no conformidades para un tipo de no conformidad cualquiera.

- Top 10 de preguntas con resultado negativo y sus causas, aquí se mostraran las diez preguntas que más veces se han contestado negativamente, en número de respuestas negativas y en porcentaje de respuestas negativas sobre el total de respuestas, es decir dos rankings distintos.

4.- Desde la parte pública se podrá acceder a diversas métricas calculadas automáticamente.

- Media de no conformidades de cada tipo por Auditoría entre dos fechas que por defecto será el último año, vendrán agrupadas por un número determinado de meses, semestres, trimestres, bimensual, mensual, etc. además existirá la posibilidad de agruparlas también por el tipo de proyecto ó proceso de la auditoría.

- Número de auditorías por mes entre fechas (por defecto por año), de la misma forma podrán agruparse los meses en bimensual, trimestral, semestral, etc.

- Número de auditorías planificadas y no realizadas entre dos fechas (por defecto por año)

5.- Desde la parte pública se podrán realizar los Seguimientos de no conformidades (Auditor), donde se contemplaran todas las no conformidades de las auditorías de un proyecto ó únicamente de la auditoría seleccionada, con la información relevante a cada no conformidad así como la posibilidad de añadir acciones correctivas / cierre, modificar el estado de la revisión, seleccionar al personal responsable, etc.

6.- Desde la parte pública se podrán modificar las fechas de la auditoría lo cual corresponde a su planificación.

7.- Antes de la ejecución de la auditoría se mostrará una agenda de la misma, mostrando la información relevante a la auditoría.

8.- La obtención de las preguntas para una auditoría se realiza de la siguiente forma:

Cada pregunta está asociada a un único proceso, y a varios tipos de auditoría, para una auditoría se obtienen todas las preguntas que están asociadas al tipo de auditoría de la auditoría seleccionada, y se ordenan por proceso.

9.- Las respuestas de una auditoría pueden ser: Si, No y No/Aplica, y podrán tener observaciones.

10.- Una auditoría puede tener los siguientes estados
- Creación: Una auditoría recién creada, sin fecha de planificación.
- Planificada: Una auditoría que tiene fecha de planificación y aun no ha sido ejecutada.
- En curso: Una auditoría que ha empezado a ejecutarse y se ha dejado a mitad, por lo que aun no está completada.
- Completada: Una auditoría que se ha ejecutado correctamente, y se han definido los tipos de no conformidad de sus no conformidades.
- Invalidada: Aquella que ha sido invalidada manualmente.
- Caducada: Es una auditoría que aun no se ha realizado y la fecha para la que se planificó ya ha pasado.

11.- Una revisión de no conformidad puede tener uno de los siguientes estados
- Pendiente: Cuando aun no se han definido sus acciones correctivas y el auditor no la ha revisado.
- En curso: Cuando el auditor ha planificado la misma y los responsables están llevando a cabo las medidas correspondientes.
- Invalida: Cuando no es necesaria su corrección por diversas causas.
- Cerrada: Cuando se ha completado su corrección.

12.- Las preguntas serán históricas, es decir al modificarse una pregunta, se dará de baja (lógica) la pregunta y se dará de alta una nueva, así las auditorías anteriores a esta modificación mostrarán la pregunta dada de baja mientras que las nuevas auditorías usaran la pregunta nueva.

13.- Del mismo modo que las preguntas los procesos serán históricos, solo que su modificación no afectará más que al nombre por lo que no requerirá baja, sin embargo si se podrán dar de baja los procesos de forma manual.

Requisitos funcionales parte privada

1.- Se podrá gestionar al personal, buscando por código de recurso a través del servicio Web.

2.- Se permitirá gestionar los procesos a auditar, pueden sufrir cambios.

3.- Se facilitará la gestión de las preguntas existentes, pudiendo cambiar ó eliminar preguntas.

4.- Todos los parámetros configurables numéricos serán configurables desde la administración, a la que se accederá mediante un login/password almacenado en la propia base de datos.

5.- Se podrán gestionar los tipos de auditoría existente permitiendo validar o invalidar los existentes y configurando si es especial o no.

Requisitos no funcionales

1.- El nuevo aplicativo resultará claro y sencillo, y su operativa será lo más amigable, intuitiva y atractiva posible.

2.- Se accederá a cualquier parte de la administración mediante login/password, manteniendo en lo posible la seguridad.

Diseño de la base de datos

Modelo conceptual

Dentro de este apartado se identifican todas las necesidades de información de cada uno de los procesos que conforman el sistema de información, con el fin de obtener un modelo de datos que contemple todas las entidades, relaciones y atributos necesarios para dar respuesta a dichas necesidades.

Hemos realizado un esquema conceptual entidad-relación de los datos, durante los rediseños hubo muchos cambios y varias versiones del esquema.

Así mismo hay datos no relacionados con el esquema pero que requieren ser guardados, son por ejemplo los diversos parámetros configurables de la aplicación, las cuentas de administración y los avisos.

Las entidades más importantes del sistema son entre otras:
La Auditoría, dado que el objetivo principal del sistema es gestionar auditorías.
La “NoConformidad”, que son las respuestas contestadas negativamente y que se revisan en un seguimiento, ya que se tiene especial interés en éstas para poder solucionar los problemas que se manifiestan por esa no conformidad.
El EstadoGeneral y el TipoNoConformidad que se explican más adelante en el apartado de implementación.

Los atributos han sido omitidos para mayor claridad y se detallan en el anexo correspondiente al diseño de la base de datos.

Los detalles se pueden ver en el anexo de diseño de base de datos (Anexo 4), donde se muestran todos los pasos seguidos hasta la obtención del esquema fisico final.

La versión final del esquema conceptual es el siguiente (Entidades relacionadas)

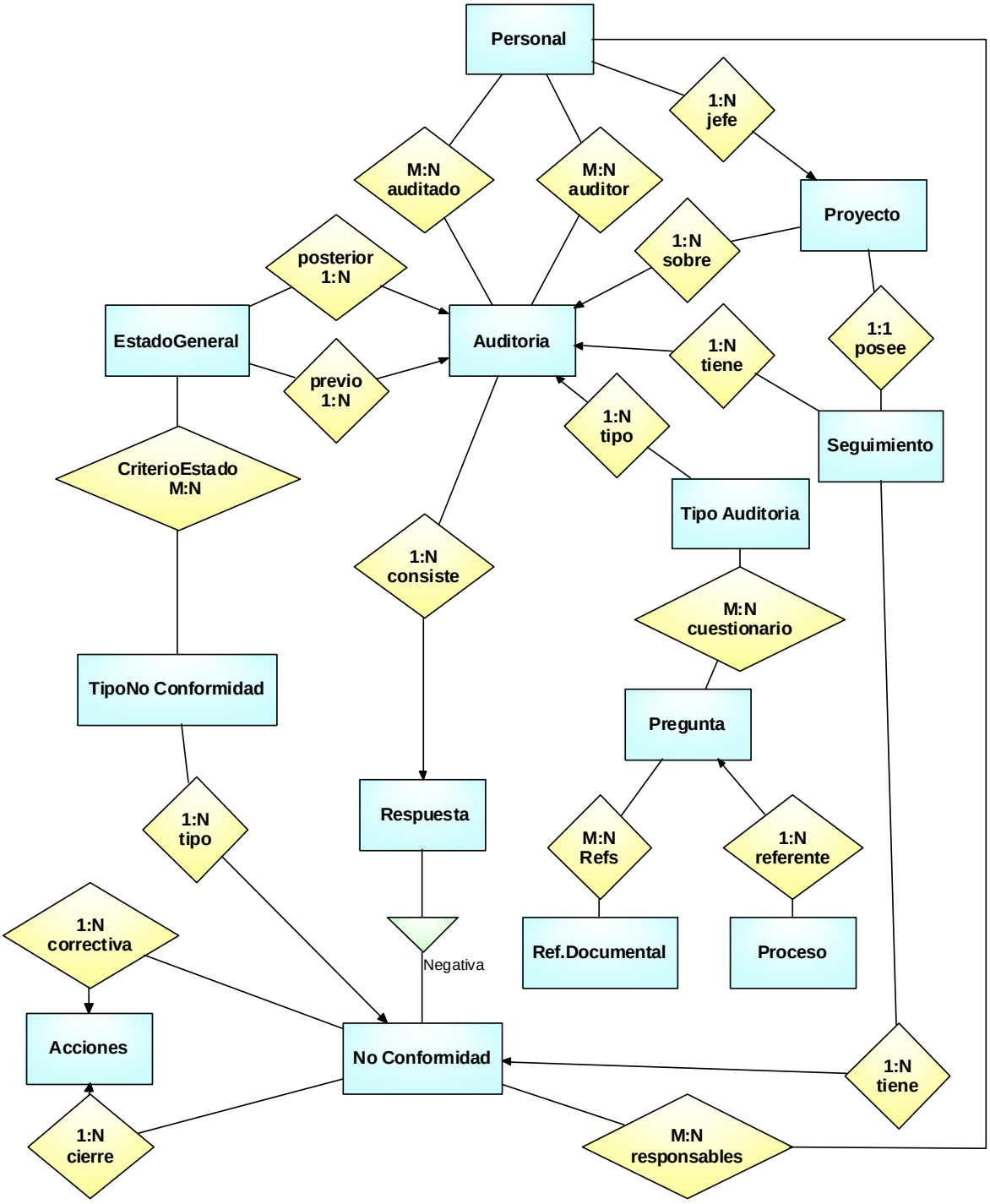


Ilustración 2 – Diseño conceptual de la Base de datos

Leyenda

Para las relaciones (1 : N), la punta de la flecha indica la entidad a la que se arrastra la clave primaria como foránea, es decir:

[Entidad A] ----- 1 : N ---> [Entidad B]: *Significa que una instancia de la entidad A puede tener relación con 0 o más instancias de la entidad B.*

Modelo lógico

A partir del modelo conceptual generado y siguiendo una serie de pautas[16], se transformó a un modelo lógico, también conocido como esquema relacional, en el cual se generan las distintas relaciones que en el diseño físico se transformaran finalmente en tablas.

Durante esta fase se ha realizado la normalización de la base de datos hasta la tercera forma normal, en este procedimiento hemos obtenido como resultado la eliminación de diversos atributos y la consideración de eliminación de otros, cuya decisión se toma durante el diseño físico.
Detallamos los aspectos más relevantes de la normalización.

Normalización de auditoría

En la entidad Auditoría disponíamos de un atributo "**valida**" que indicaba la validez o invalidez de la auditoría, y otro atributo "**estado**" que marcaba el estado en el que se encontraba la auditoría (en curso, planificada, caducada, finalizada, invalidada).

En la segunda forma normal encontramos la dependencia funcional entre el atributo estado y válida.

Si "**estado**" era invalidada, entonces "**valida**" era falso.
Si "**estado**" era distinto de invalidada, entonces "**valida**" era verdadero.

Se tomó la decisión de eliminar el atributo "**valida**", dado que podía inferirse del atributo "**estado**".

Normalización de Proceso

En la normalización de Tipo de Proceso, el atributo "nombre" que corresponde al nombre del proceso era un simple atributo, sin embargo se vio que era clave candidata.

Modelo físico

Durante el diseño físico se tomaron decisiones para aumentar el rendimiento de la base de datos, a continuación detallamos algunas de las decisiones tomadas durante esta fase.

Claves candidatas de gran tamaño

Para tablas con claves candidatas grandes, donde teníamos por ejemplo en la entidad pregunta una clave candidata que era la propia pregunta (un texto de tamaño 300), se ha optado por crear una clave autogenerada mucho más pequeña de forma que la indexación fuera más eficiente.

Claves modificables

En la entidad Tipo de Proceso, había una clave candidata "Nombre" que el sistema debía permitir cambiar, este cambio afectaría a todas sus relaciones en forma de cascada, sin embargo sabemos que SQL Server 2005 tiene problemas en la

actualización en cascada, por lo que hemos decidido ponerlo como atributo normal y crearle una clave primaria autogenerada.

Referencia documental

Teníamos una entidad Referencia documental, donde sólo necesitábamos conocer el nombre de la referencia documental, lo cual hace indicar la poca importancia de esta entidad.

Esta entidad tenía una relación M:N con la entidad "**Pregunta**", lo cual nos llevaba a obtener dos tablas, una contenida en otra:

fk: Clave foránea
pk: Clave primaria

Referencia_Documental__Pregunta (nombre_documento (fk), referencia_pregunta (fk))
Referencias_Documentales (nombre_documento (pk))

En la tabla de la relación (Referencia_Documental_Pregunta) aparecen todas las referencias documentales de la tabla Referencias_Documentales, por lo que ésta segunda tabla se podía obtener de la primera, lo cual hizo la posibilidad de eliminar dicha tabla

Sin embargo, al estar utilizando un mapeador automático a relaciones, esta tabla tubo que mantenerse, dado que para dicha relación, JPA (estándar de mapeo objeto-relacional) se tiene en la base de datos tres tablas, una para cada entidad y otra para la relación, y no existe la posibilidad de eliminar ninguna de ellas.

El esquema físico final puede encontrarse en el anexo del diseño de la base de datos.

Diccionario de datos

Hemos generado un diccionario de datos en el cual se definen todos los atributos del esquema físico final para aclarar su significado y su comportamiento en el sistema a desarrollar, de forma que en caso de futuras ampliaciones, mejoras o mantenimiento del sistema, se pueda acceder a una definición aclaratoria de cada uno de los atributos del modelo de datos.

Diseño del sistema

El diseño del sistema consiste en definir cada uno de los apartados de los que se compone el sistema así como el comportamiento y las decisiones de diseño adoptadas para obtener una solución lógica abarcando todos los requisitos del sistema.

Además en el diseño se generan diversos esquemas que dividen en diversas capas el sistema con el fin de facilitar la comprensión del mismo.

Así mismo durante esta fase se eligen los patrones de diseño más adecuados para implementar el sistema y que se adecuan a las necesidades requeridas que se recogieron durante la fase de análisis.

Arquitectura software

La arquitectura del sistema se ha dividido en tres capas, siguiendo un patrón de diseño de arquitectura software para aplicaciones Web (Model View Controller)[17], que son el Modelo, la Vista y el Control.

Para facilitar esta tarea hemos utilizado Spring Framework[14] que implementa este patrón, y facilita la implementación de la aplicación Web ofreciendo múltiples herramientas, como por ejemplo seguridad para el acceso y mantenimiento de sesiones.

Al tratarse de una aplicación Web, el esquema parte de un navegador Web desde el cual el usuario podrá acceder a la aplicación.

En la siguiente figura podemos observar el esquema general de la aplicación sin entrar mucho en el detalle físico.

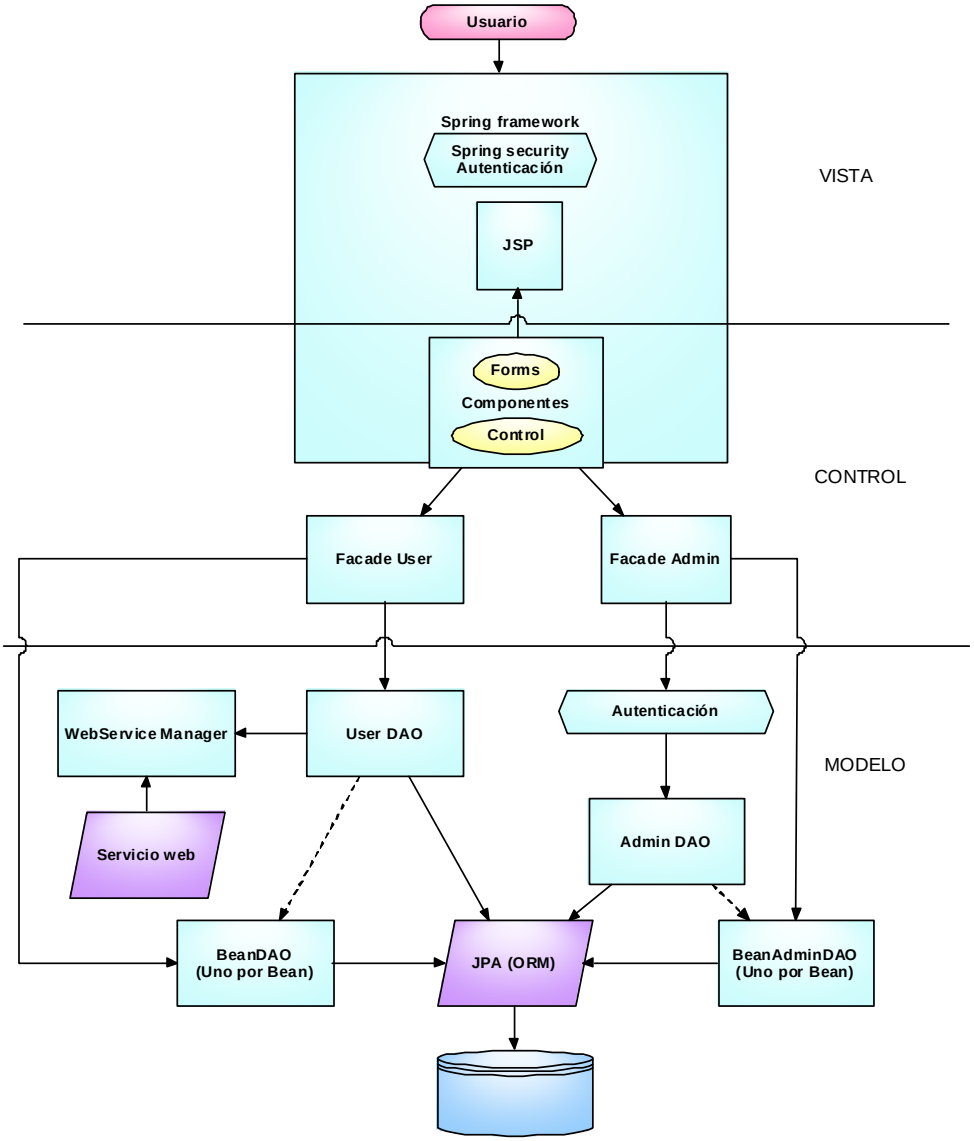


Ilustración 3 - Diagrama de Componentes Software

Capa de Modelo

La gestión del mapeo objeto-relacional se delega en manos de JPA, reconocido estándar que cuenta con diversas implementaciones, por lo que resultará bastante fácil y rápido el cambio de proveedor.

Se utilizan dos formas de acceso a datos, una para usuarios y otra para administradores, en la segunda se requiere autenticación que irá contra la base de datos donde las contraseñas se almacenan cifradas.

Todas las operaciones de administración requieren estar autenticado, de forma que saltarse la autenticación en la capa de Vista (Autenticación de Spring security[15]) no otorgará los permisos para acceder a los datos, solo a la capa de vista de la administración, pero vacía.

De esta forma se tienen dos niveles de seguridad, uno en la capa de vista y otro en la capa del modelo.

Capa de control

La capa de control está formada por diversos componentes, llamados por Spring Framework, que son los que controlan la gestión de los datos requeridos por la página solicitada (vista), de forma que acceden a los datos a través de la fachada correspondiente.

La lógica de negocio está implementada en estas acciones o controladores, parte de la funcionalidad también está implementada en la fachada que ofrece diversos métodos más completos.

Las dos fachadas proveen a los controladores acceso a lectura y modificación de datos, de una forma más o menos simplificada. De esta forma se encapsula el acceso a datos de la lógica de negocio y además de separar la parte pública de la privada.

Capa de vista

La implementación del software de servicio en el sistema central se basa en la utilización de un Servidor de Aplicaciones. Entre otros, el Servidor de Aplicaciones ofrece los siguientes recursos fundamentales:

Servidor Web – Capaz de servir páginas Web de construcción dinámica.

Seguridad – Ofrecerá un sistema de autenticación para permitir la visualización de diversas vistas de administración, esta funcionalidad se delega en manos de Spring Security[15].

Componentes – Sistema capaz de ejecutar componentes de software que incluya mecanismos estándar para soportar la concurrencia y controlar las distintas sesiones de trabajo

En la capa de vista se implementan las distintas vistas de la aplicación, que corresponden a las páginas JSP, que tras pasar por el servidor de aplicaciones se transforman en una página HTML dinámica.

Patrones de diseño

Los patrones de diseño son formas recomendables de implementar aspectos comunes a diversas aplicaciones, se han diseñado para ser fácilmente implementables, reutilizables y mantenibles, de forma que resulten ser una buena solución.

Además del patrón MVC[17] explicado en el apartado anterior que corresponde a un patrón de diseño de la arquitectura general, hemos adoptado otros que hemos considerado interesantes para nuestra aplicación.

Patrón Data Access Object (DAO)

El primero de los patrones utilizados es el patrón DAO[18] para la capa de Modelo de datos, donde se encapsula la forma de acceso a los datos.

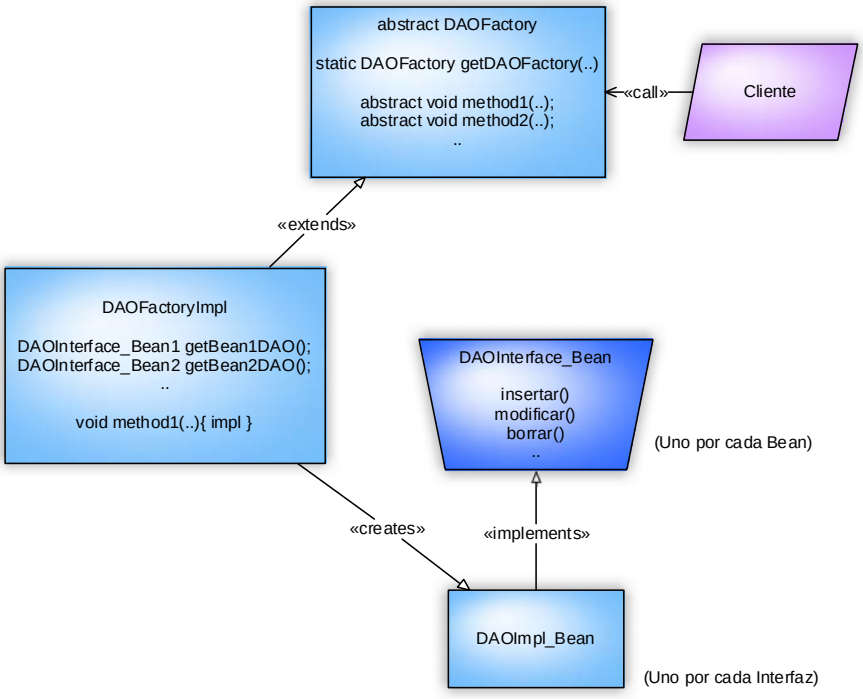


Figura 1 – Patrón DAO

De esta forma, el cliente lo único que conocerá de la capa del modelo, serán los métodos ofrecidos por la clase DAOFactory, sin llegar a conocer la implementación propia de los métodos, incluso podríamos tener varías implementaciones del mismo

método para distintas bases de datos o distintas formas de implementarlo (mediante algún ORM, de forma manual con sentencias Standard Query Language (SQL), etc.).

En nuestro caso, el cliente es la fachada correspondiente (explicadas más adelante), que utiliza tanto el DAOFactory, como los diversos DAOInterface_Bean que crea el DAOFactory.

Patrón Fachada

Este patrón lo hemos utilizado para abstraernos aun más de la jerarquía utilizada para el acceso a datos (la del patrón DAO), mientras que de esta forma una única clase nos provee de todos los métodos necesarios y además se encarga de facilitarnos algunos métodos algo más complejos que incorporan cierta lógica de negocio, como por ejemplo la obtención del cuestionario de preguntas para una auditoría.
Además tendremos implementadas dos fachadas diferentes, una para el acceso a datos como usuario y otra para el acceso a datos como administrador, de esta forma encapsulamos en cada fachada distinta funcionalidad y además simplificamos el uso de los DAO.

Diseño de clases

En el diseño de clases transformamos los requerimientos del sistema, tanto funcionales como de almacenamiento de datos en un esquema de clases donde se describen las clases principales y los métodos necesarios para su manejo.

Los métodos descritos hacia el acceso a datos se han especificado tras el diseño de los casos de uso, donde se observa con más claridad qué datos hacen falta para cada apartado, de forma que serán los métodos a implementar por las fachadas.

El diseño de clases final es muy similar al diseño físico de los datos, dado que el mapeo de JPA es una aproximación a una Tabla por Clase.

En la siguiente figura podemos observar el diseño de clases realizado.

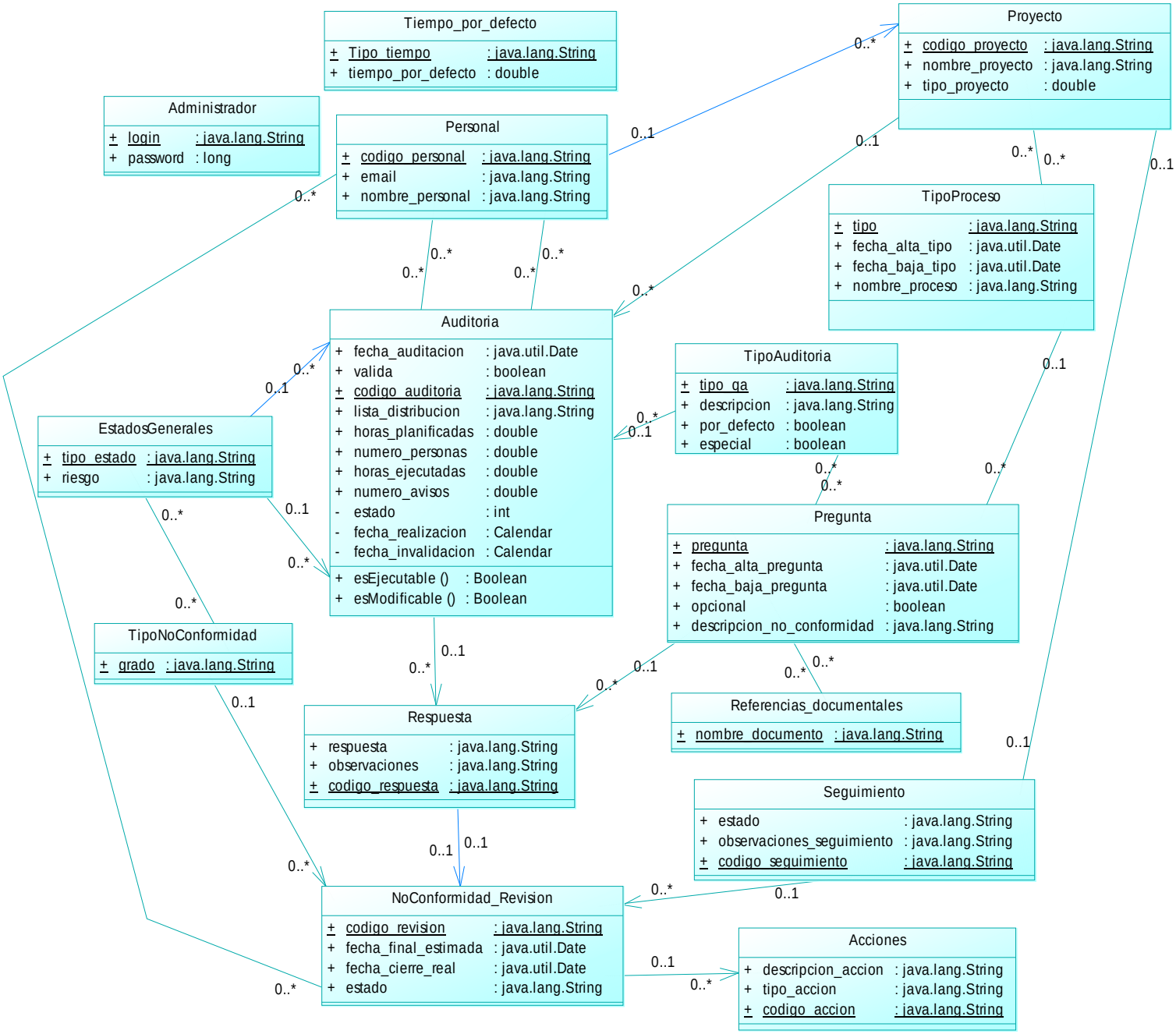


Ilustración 4 – Diagrama de clases

Casos de uso

Con el fin de obtener unos requisitos refinados y un esquema de los procesos que tienen lugar en cada apartado (módulo), hemos realizado el diseño de los casos de uso del sistema.

De este proceso obtenemos los distintos apartados a desarrollar, así como los métodos necesarios de la capa de datos para acceder a los datos.

Destacamos los más relevantes que tienen que ver con la gestión de las auditorías (Creación, modificación y ejecución de las mismas).

Gestión de auditorías.

Inserción

RF3.- *INSERCIÓN* Mostrar proyectos existentes
RF3.- *INSERCIÓN* Añadir proyectos por código de proyecto obteniéndolos mediante el servicio Web.
RF3.- *INSERCIÓN* Mostrar personal existente.
RF3.- *INSERCIÓN* Añadir personal por código de recurso obteniéndolo mediante el servicio Web.
RF3.- *INSERCIÓN* Posibilidad de seleccionar varios como auditores y varios como auditados.
RF3.- *INSERCIÓN* Mostrar los tipos de auditoría válidas existentes, permitiendo seleccionar varias especiales o varias no especiales.

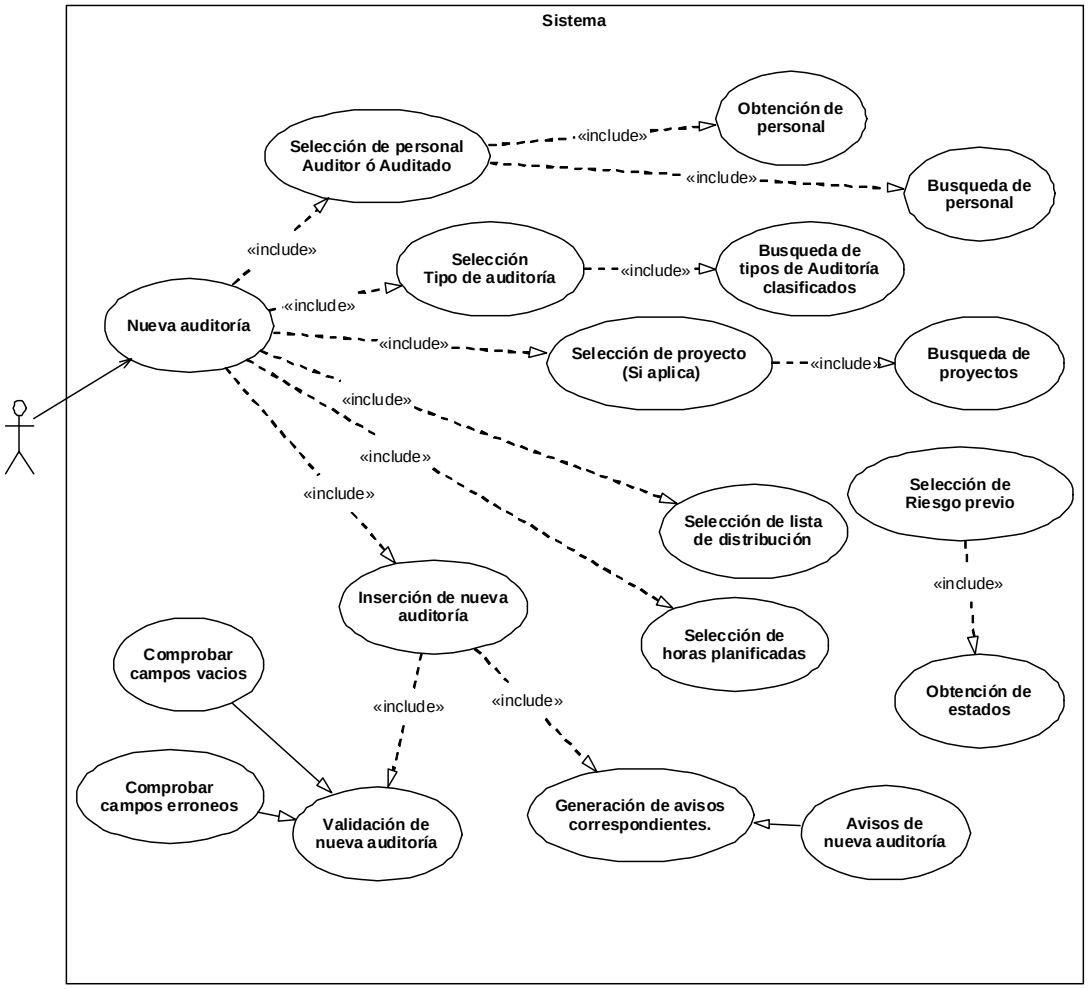


Ilustración 5 – Caso de uso inserción

Como podemos observar se parte de que el usuario quiere crear una nueva auditoría, y se definen todos los pasos que debe realizar, es decir seleccionar cada una de las opciones necesarias y requeridas para la auditoría.

Se parte de un apartado (Nueva Auditoría en el ejemplo) y los requisitos funcionales que se deben tener en cuenta en este apartado (RF3 en el ejemplo).

Los procesos finales son aquellos que se requieren en el acceso a datos, como son: búsqueda de personal, búsqueda de tipos de Auditoría clasificados, búsqueda de proyectos y Generación de avisos correspondientes. Además de nuevos apartados (como por ejemplo "*poder añadir proyectos obtenidos mediante servicio Web"*) y también aspectos a tener en cuenta en la vista y control ("*posibilidad de seleccionar varios como auditores y varios como auditados")*.

Los procesos intermedios son aquellos que se deben tener en cuenta en la capa de control para poder realizarlos, así como en la vista para poder tener acceso a ellos.

Contra más detallado sea un caso de uso, menos dejaremos para el refinamiento del diseño en la implementación.

Modificación

RF3.- *MODIFICACION* Mostrar información referente a una auditoría permitiendo modificarla.
RF3.- *MODIFICACIÓN* Posibilidad de invalidar auditorías.

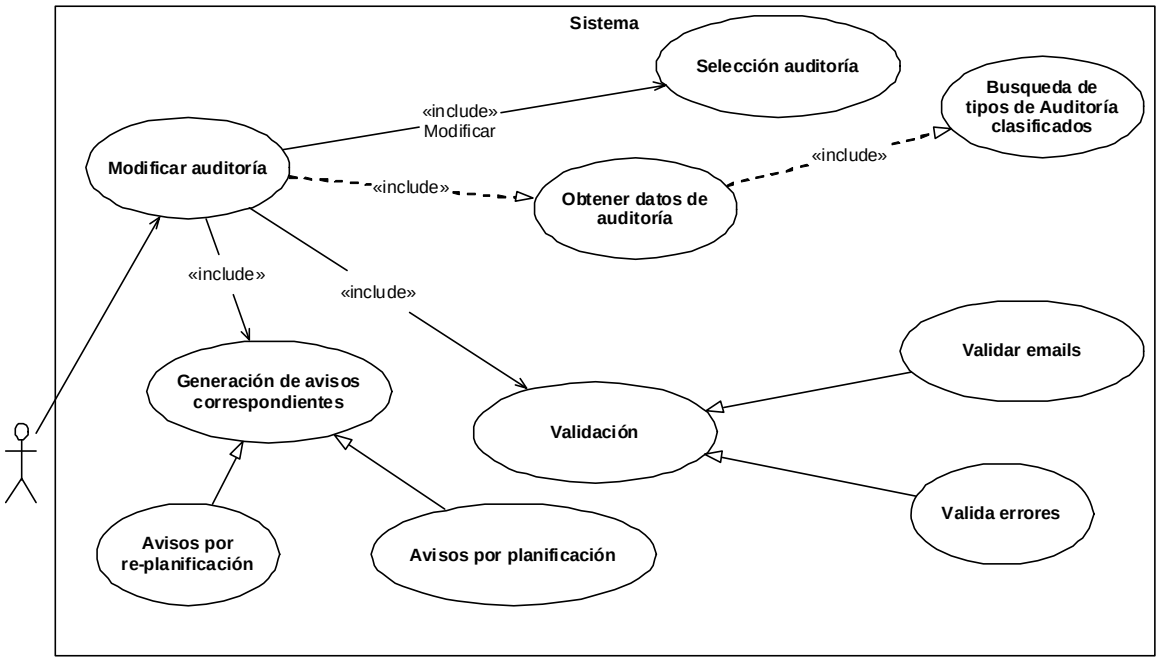


Ilustración 6– Caso de uso modificación

En este caso de uso podemos observar que la mecánica es similar, obtención de datos, utilizar esos datos, validar y generar los avisos correspondientes, en este caso debidos a la planificación de la auditoría o replanificación.

Ejecución

RF3.- *EJECUCIÓN* Acceso al cuestionario (si existe), generación del cuestionario, ordenándolo por proceso y mostrando aquella página (preguntas de un proceso) seleccionado.
RF3.- *EJECUCIÓN* Saber cuantos procesos tienen lugar en la ejecución para la generación de las pestañas.

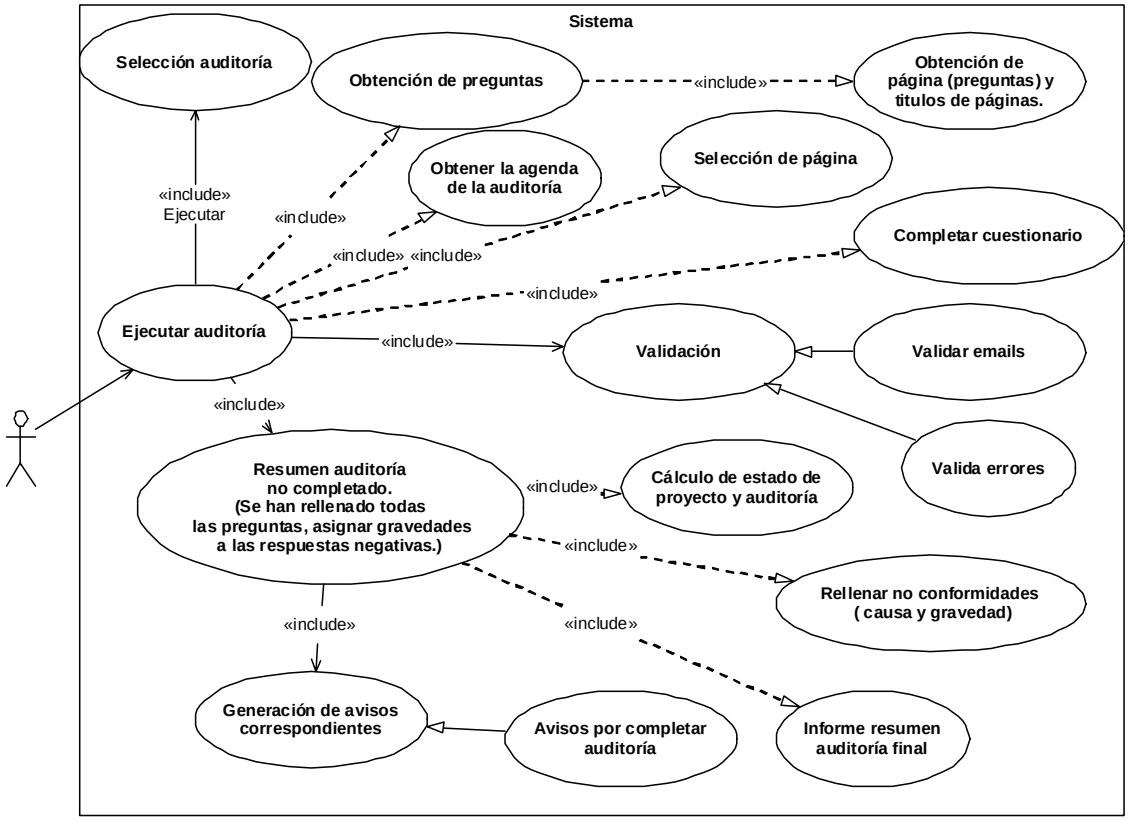


Ilustración 7 – Caso de uso ejecución

Observamos que en la ejecución se ha añadido un requisito funcional no especificado en el documento inicial de análisis, la selección de página, dado que la ejecución de la auditoría se desea que sea de forma paginada, en lugar de mostrar todas las preguntas en una sola vista.

Además en este caso de uso se ha añadido un subcaso de uso (resumen de auditoría) dado que forma parte de la ejecución de la auditoría, pese a existir una diferenciación clara entre el cuestionario y el resumen final.

Implementación

Durante la fase de implementación han surgido diversos problemas a los cuales hemos aplicado distintas soluciones que explicaremos a continuación.

La metodología empleada en el desarrollo consiste en:

- Refinar el análisis y diseño previo a un apartado
- Implementar dicho apartado
- Realizar las pruebas unitarias
- Integrarlo al sistema
- Realizar las pruebas de integración

Así pues se siguieron estos pasos durante el desarrollo de cada uno de los apartados correspondientes intentando seguir el calendario fijado estimando el tiempo que nos llevaría realizar todo el sistema completo.

Los problemas más importantes e inesperados con los que nos hemos encontrado han retrasado un poco el desarrollo del sistema.

MultiActionFormController

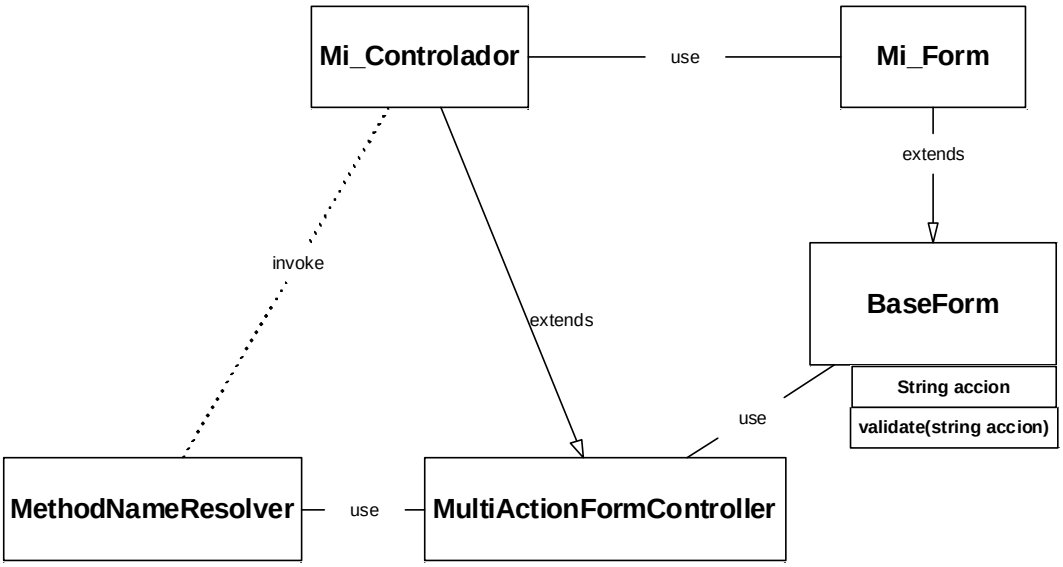
Problema

El Framework utilizado no dispone de una forma sencilla de tener una vista formulario con múltiples acciones, no ocurre así con una vista normal (que no sea formulario) con múltiples acciones (conocida como MultiActionController[19]), por lo que decidimos implementarnos nuestro propio controlador base, que facilitara la implementación de un controlador para formularios donde se puedan ejecutar diversas acciones.

Solución

Hemos diseñado unas clases base de forma que se pueda implementar este controlador para que puedan invocarse los distintos métodos de cada acción con parámetros.

La solución en un diagrama de clases simplificado es la siguiente:



Simplificación: Cómo simplifica el problema

El usuario implementa Mi_controlador y Mi_form manteniendo así la metodología de Spring en la que se implementa un controlador y una clase formulario. Sin embargo esta vez, extendiendo (**extends**) las clases generadas (MultiActionFormController y BaseForm).

En el controlador, se deben implementar diversos métodos (uno por acción), lo cual es la forma más cómoda de realizar un controlador múltiple para formularios.

En la vista, deberá haber un parámetro oculto con nombre “acción”, de forma que se almacenará en el BaseForm.
Por ejemplo:
<button name=”acción” value=”guardar(1&2&3)”>Guardar</button>

La clase Mi_Form que extiende BaseForm deberá implementar el método “validate(String acción)” que valida las distintas acciones que pueden ser llamadas, de esta forma se evita la posibilidad de poder llamar a cualquier método desde el parámetro oculto. Esta clase sirve para almacenar los campos del formulario siguiendo la filosofía de Spring.

Cómo funciona

El MultiActionFormController implementa el método onSubmit de Spring (Simula que es un controlador de formulario simple, con una sola acción), en este método lo que hace es obtener la acción completa del input oculto y enviárselo a la clase que lo transforma en nombres (MethodNameResolver), que se encarga de leer la acción y los parámetros para reconocer a qué método llamar (**invoke**) de la clase Mi_controlador, después de validar la acción.

Si el método no existe, el MultiActionFormController no ejecuta ninguna acción y nos devuelve a donde estábamos antes de llamarle.

Para la acción del ejemplo “guardar(1&2&3)” se ejecutar a el método:

public ModelAndView guardar(int uno, int dos, int tres, …);

Donde los puntos suspensivos son los parámetros propios de Spring (HttpServletRequest request, HttpServletResponse response, Object command, BindException errors), que se comportan de la misma forma que lo hacen en Spring Framework al realizar un controlador de formulario simple.

Servicio Web

En la implementación de la obtención de datos mediante un servicio Web, nos encontramos con un problema de interoperabilidad, debido a las distintas tecnologías utilizadas para la obtención del servicio y el servidor del servicio.

Problema

De esta forma el servicio Web estaba implementado mediante WSE 3.0, cuyo fichero de descripción de servicio web (*Web Services Description Language* WSDL[20]) no incorpora información sobre las cabeceras de seguridad requeridas por el servicio, esto se debe a que sigue un estándar diferente, y desactualizado, al que siguen los actuales servicios Web, que incorporan en el WSDL la información requerida por el servicio [21] cabeceras.

Solución

Tras mucho tiempo buscando soluciones por la Web, encontramos que el error se basaba en que las llamadas al servicio no incluían las cabeceras, así pues bastaba con añadir en la información de despliegue un resolutor de direccionamiento que añadiera las cabeceras pertinentes.

```
<handler type="java:org.apache.axis.message.addressing.handler.AddressingHandler">
        <parameter name="referencePropertyNames" value="*"/>
</handler>
```

De forma que se añadan todas las cabeceras, sean o no requeridas por el servicio Web.

La solución recomendada en muchos sitios era olvidarse de WSE 3.0 y utilizar otra tecnología que siguiera el estándar, lo cual no nos servía dado que el servicio Web ya estaba implementado y debía usarse ese.

Mapeo objeto relacional

En el mapeo objeto relacional de JPA hemos utilizado anotaciones, lo cual resulta más cómodo y sostenible que los ficheros xml propios de *Hibernate*, de esta forma podemos cambiar de proveedor tan solo cambiando en fichero en el cual se define la configuración de la persistencia el atributo del proveedor:

```
<provider>org.hibernate.ejb.HibernatePersistence</provider>
```

Problema

En este mapeo objeto-relacional nos encontramos con un problema a la hora de mapear una relación M:N con un atributo de por medio, en nuestro caso se trataba de la relación existente entre Estado y TipoNoConformidad, que tienen una relación M:N con un atributo “maximo_numero_no_conformidades”.

Solución

La solución en objetos del mapeo es crear una clase que corresponda a la relación, en nuestro caso la hemos llamado CriterioEstado. Como la clave primaria de esta relación es una clave compuesta, hemos tenido que crear otra clase llamada CriterioEstadoPK que corresponde a la clave primaria de la relación, que posee un atributo por cada parte de la clave, como la clave de CriterioEstado está compuesta por la clave primaria de Estado y la clave primaria TipoNoConformidad, tenemos en la clase CriterioEstadoPK ambos atributos de dicho tipo.

```
@Column(name="tipo_no_conformidad")
private String  tipo_no_conformidad;
@Column(name="estado")
private String estado;
```

Y en la clase CriterioEstado, el atributo clave embebido (en la clase CriterioEstadoPK):

```
@EmbeddedId
protected CriterioEstadoPK primarykey;
```

Esta solución es muy farragosa y poco usable desde el punto de vista de orientación a objetos, sin embargo es una de las dos soluciones propuestas en JPA para este caso, y es la única que funcionaba en Hibernate.

Carga perezosa

Problema

Otro de los problemas comunes en el uso de JPA como motor de persistencia, son los atributos cargados de forma perezosa (Lazy), lo cual significa que no se obtienen directamente los datos de un objeto cuando se realiza la propia consulta, sino cuando se accede a realmente a los datos.

Esto ocurre con listas de objetos cuando la relación es 1:N o M:N, por ejemplo, cargar todas las respuestas de una auditoría cuando cargamos un objeto Auditoría, posiblemente no accedamos a ninguna de las respuestas ya que sólo nos interesa saber la fecha de la auditoría, por lo que cargar siempre los datos es mucho más costoso, sin embargo mantener en la cache de Hibernate, todos los objetos Lazy que deben ser cargados en caso de consulta, también es algo costoso de mantener.

Solución

El mantenimiento en la cache de los datos que están siendo consultados con la opción Lazy, deben estar disponibles desde la llamada al controlador (donde se consultan los datos por primera vez), hasta que se carga completamente la página dinámicamente (jsp), por lo que al inicio de la ejecución de un controlador, es posible limpiar la cache de todos estos datos (anteriores) ya que no van a volver a usarse.

Paginación

La paginación es una forma de visualización de datos muy extensos donde se divide la información en diversas páginas, en nuestro caso el cliente nos pidió expresamente la paginación de las preguntas por proceso, tanto en las auditorías como a la hora de gestionar las preguntas.

Problema

La paginación puede llegar a ser muy ineficiente. Puede resolverse de dos formas distintas, enviar todos los datos al cliente y que el cliente realice la paginación, o filtrar los datos en el servidor y enviar solo los datos referentes a la página.

Solución

Nosotros hemos escogido la paginación en el servidor, por ser la forma más limpia, eficiente y mantenible.

Para aumentar la mantenibilidad, hemos decidido separar la paginación en una clase que ofrezca métodos de paginación, donde la forma de implementarla pueda cambiarse en un futuro de forma cómoda.

Hemos creado una clase abstracta y genérica Paginator la cual extienden las diversas clases que pueden paginar un tipo de elemento en páginas de algún tipo. Paginator<TypeElem, TypePage>, para ordenar por ejemplo, en nuestro caso particular, Respuestas por Proceso y Preguntas por Proceso.

En esta clase Paginator hay un método abstracto en el cual a partir de todos los elementos a paginar devuelve aquellos referentes a la página solicitada así como los nombres de las diferentes páginas seleccionables.

Sugerencias

Las sugerencias son ayudas que el sistema ofrece al usuario en la selección de un objeto de un listado previsiblemente grande, como por ejemplo el personal de la empresa en la selección de un Auditor, auditado o responsable.

Un ejemplo de sugerencia lo encontramos en el motor de búsqueda de Google, que nos ofrece diversas sugerencias a la vez que nosotros escribimos.

Problema

Al crear una auditoría, se pide al usuario seleccionar tanto al grupo de auditores como auditados. Es posible que el número de personal de la empresa almacenado en la base de datos sea muy elevado, con la consecuencia directa de la incomodidad de seleccionar al personal.

Solución

Por ello se planteo la solución de implementar *Ajax Suggestion*[22], lo cual permitiría una mejor usabilidad.

Sub-problema

Las implementaciones de "*Ajax sugestión*" encontradas eran poco genéricas, por lo que se realizo una generalización de una de ellas para poder usarlas en cualquier parte de la aplicación Web, así como poder aplicarla múltiples veces en una única vista, realizando diversas operaciones y manteniendo la sesión del formulario de Spring.

Avisos

Los avisos que el sistema debe generar, son emails que se envían en distintos periodos de tiempo y debidos a la realización de alguna tarea en la aplicación.

Problema

El problema principal de la gestión de avisos consiste en que una sola tarea, por ejemplo la planificación de una auditoría, genera múltiples avisos a múltiples personas y en periodos de tiempo distintos, por lo que se requiere de algún proceso que compruebe la fecha y envíe los avisos que correspondan.

Solución

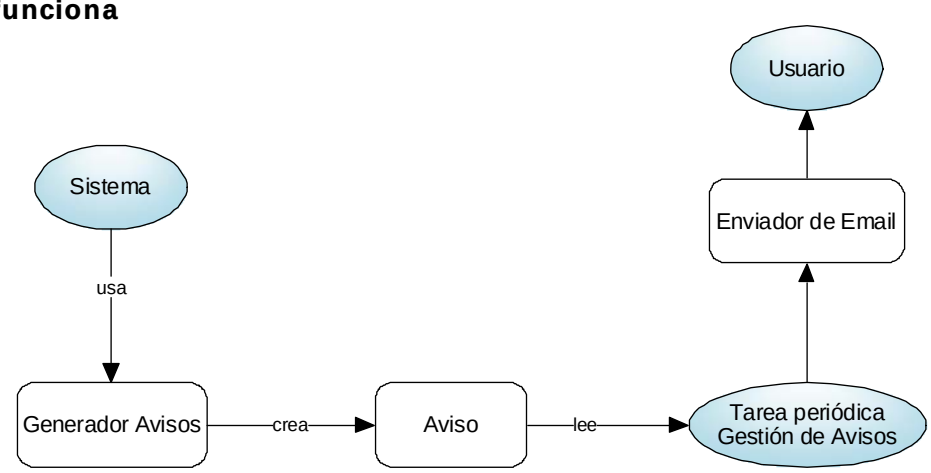
La solución adoptada trata de la creación de un único servlet, que se ejecuta de forma única en el servidor y que contiene diversas tareas periódicas que se ejecutan una vez al día en horario fuera de oficina.

Para ello hemos utilizado TimerTask[23], que nos facilita la tarea de la periodicidad, asignándole dos tareas periódicas que requiere el sistema, una es la de enviar avisos pendientes y la otra el caducar las auditorías para las que se ha pasado su fecha planificada para ejecutarla.

Así pues sólo necesitamos almacenar los avisos pendientes para una fecha determinada, existiendo diversos tipos de aviso (cada tipo de aviso enviará una información diferente), y un email al cual enviárselo (en caso de no existir email, se envía el aviso a toda la lista de distribución).

Ésto modificó el diseño de clases y de la base de datos inicial, añadiendo una nueva entidad *Aviso*, que no habíamos tenido en cuenta.

Cómo funciona



El sistema crea los avisos a través del Generador de Avisos (siempre a través de la fachada).

La tarea periódica lee los avisos pendientes, genera los mails y los envía utilizando el módulo encargado de enviar mails.

Estado del proyecto y tipo de no conformidad

Problema

Uno de los requisitos del sistema era dotarlo de la máxima configuración posible, por ello la aplicación debe permitir configurar la forma de calcular el estado de un proyecto ó auditoría basándose en el número de respuestas negativas y su gravedad.

Solución

La solución que hemos adoptado ha sido el poder administrar las gravedades de las no conformidades (Tipo de no conformidad) y los nombres para los estados de un proyecto dependiendo del número de no conformidades (Estado General).

De tal forma que exista una fórmula de calculo entre el número de no conformidades y un estado general.

De esta forma que se define una función que describa la pertenencia a un estado dependiendo del número de no conformidades de un tipo. Siendo ambos, tanto el estado como la gravedad de las no conformidades configurables, así como todas las constantes.

Se define una función por cada tipo de no conformidad de la siguiente forma:

$$f(x)=\begin{cases} estado_1 \xleftarrow{Si} x \le const_1 \\ estado_2 \xleftarrow{Si} const_1 < x \le const_2 \\ \dots \\ estado_n \xleftarrow{Si} const_{n-1} < x \le const_n \end{cases}$$

Donde ***x*** es el número de no conformidades de cierta gravedad (tipo de no conformidad).

Las constantes $const_1$, $const_2$, … , $const_n$ así como los estados $estado_1$, $estado_2$, … , $estado_n$ y las gravedades de las no conformidades deben ser configurables.

La solución es, tener al Estado relacionado con cardinalidad M:N con TipoNoConformidad, con un atributo en la relación *"maximo numero no conformidades"*, que corresponde a la constante $const_i$ de la fórmula.

Lo explicamos con un ejemplo, imaginemos que los estados son Bueno, Regular y Malo, y las gravedades son Leve, Grave.

Malo si hay 6 o **más** no conformidades **Graves(s)**
Regular si hay entre 1 y **5** no conformidad **Grave(s)**

Bueno si hay hasta **0** no conformidad **Grave(s)**
Malo si hay 10 o **más** no conformidades **Leve(s)**
Regular si hay entre 3 y **9** no conformidades **Leve(s)**
Bueno si hay hasta **3** no conformidad **Leve(s)**

Estas relaciones o fórmulas corresponderían a las relaciones:

Malo	Inf.	Grave
Regular	5	Grave
Bueno	0	Grave
Malo	Inf.	Leve
Regular	9	Leve
Bueno	3	Leve

Tabla 1 – Implementación de la fórmula de cálculo

De esta forma tenemos que tanto la fórmula, como los tipos de no conformidad como los estados son configurables, dado que se establece una relación entre todos los tipos de no conformidad y todos los estados.

Pruebas

Introducción

Se han realizado varios tipos de pruebas para los cuales hemos utilizado una herramienta en la cual se planifican las pruebas, se ejecutan y se obtienen resultados y métricas a raíz de la ejecución. La herramienta utilizada se llama *Testlink*[9].

Con esta herramienta lo que conseguimos es aislar la planificación de la ejecución, por lo que distintas personas planifican y ejecutan las pruebas de forma independiente.

Plan de pruebas

Hemos realizado pruebas de validación, robustez, usabilidad y aceptación, así como pruebas unitarias y de integración durante el desarrollo del producto.

El proceso que seguimos en la planificación de las pruebas fue diferente dependiendo del tipo de pruebas.

Unitarias

Estas pruebas se utilizan para comprobar el correcto funcionamiento de un modulo del sistema, su realización se produce en el momento justo posterior a la implementación del modulo, y una vez el módulo está completamente probado y corregido, se integra en el sistema realizando las pruebas de integración.

Integración

Las pruebas de integración se realizan durante el desarrollo, acoplando una unidad (modulo), que se ha probado de forma unitaria y que cumple con alguno de los apartados, al sistema, de forma que no produzca efectos colaterales y el resto del sistema siga funcionando igual que hasta ahora.

Validación

Para las pruebas de validación se partió de los requisitos, generando uno o más casos de prueba por requisito, cubriendo toda la funcionalidad. El objetivo de estas pruebas es comprobar que la aplicación cubre toda la funcionalidad recogida en los requisitos obtenidos durante el análisis

Robustez

En las pruebas de robustez se intenta hacer que el sistema falle, introduciendo todo tipo de datos incorrectos o no esperados. El objetivo de estas pruebas es solucionar todos los posibles errores que pudieran aparecer a la hora de interactuar con el sistema.

Usabilidad

En las pruebas de usabilidad se establecen diversas metas, es decir, se le pide a un usuario inexperto que realice diversas tareas o alcance diversos puntos de la Web, guiándole lo menos posible, de forma que sea el propio usuario quien deba buscarlas. El objetivo de estas pruebas es comprobar cómo ven los usuarios cada apartado de la Web (dónde miran, dónde entran,…) al pedirle cumplir unos objetivos.

Aceptación

Las pruebas de aceptación consisten en poner al cliente final delante de la aplicación, para que pruebe todos los aspectos del sistema y realice un tipo de validación, dónde se comentan las funcionalidades pendientes ó aspectos que sólo el cliente conoce debido a un mal análisis. El objetivo de estas pruebas es que el cliente acepte el sistema final o solicite cierta funcionalidad extra ó pendiente de realizar.

Resultado de las pruebas

En el anexo de resultado de pruebas pueden observarse los resultados detallados de cada caso de prueba, habiendo realizado la empresa otros casos de prueba generales para aplicaciones destinadas a cliente final, que no se recogen en este documento dado que no estan a nuestra disposición.

Unitarias y de integración

Las pruebas unitarias y de integración resolvieron multitud de problemas menores de implementación, que son difícilmente recogidos en un documento de resultados.

Para estas pruebas se implementaron clases de prueba con casos de prueba utilizando la tecnología JUnit[24], que nos permite repetir las pruebas unitarias una y otra vez desde el entorno de desarrollo, lo cual permite realizar las pruebas unitarias sobre el sistema integrado, lo cual nos asegura (hasta cierto punto), que no se producen efectos colaterales en el resto de unidades lógicas.

Esta tecnología nos permite realizar aserciones de igualdad, de veracidad, de falsedad etc., de forma que se comprueben que los resultados devueltos por los módulos son los esperados.

Validación

Las pruebas de validación obtuvieron unos resultados en general positivos, por lo que no sirvieron de mucho, se encontraron algunos errores menores, sin embargo se satisfacía toda la funcionalidad requerida por el sistema.

Esto se debe a que la forma de desarrollar se basaba en la creación de módulos (apartados) que cumplían con alguna funcionalidad de un requisito, siguiendo ordenadamente los requisitos, es decir el desarrollo se basaba en los requisitos.

Robustez

Durante la ejecución de estas pruebas se encontraron multitud de errores de verificación de los datos y parámetros, así como el correcto manejo de errores. En la mayoría de los casos no se avisaba al usuario de un error (por ejemplo, introducción una fecha incorrecta, o número incorrecto, email no válido, etc.).

Durante estas pruebas se localizaron también aspectos de visualización, donde algún apartado de la Web no tenía un aspecto similar al resto.

Usabilidad

Gracias a estas pruebas se ha visto como actúa el usuario ante el sistema desconociéndolo y sin ser informático, de esta forma se comprobó que algunas cosas no se entendían bien, resultaban incomodas o incomprensibles y otras no sabía como hacerse.

Por ejemplo, en la ejecución de una auditoría, el sistema terminaba la ejecución cuando se habían completado todas las preguntas, sin embargo el usuario quería cambiar una de las respuestas para añadir observaciones y el sistema no se lo permitía.

En seis de las dieciséis pruebas realizadas se encontraron problemas en la comprensión del sistema.

Aceptación

Lamentablemente estas pruebas no pudieron ser ejecutadas, debido a que el cliente final no disponía de tiempo material para ejecutarlas.

Conclusiones

La experiencia del trabajo en la empresa ha sido fructífera dado que hemos aprendido a utilizar las herramientas de forma profesional, ya que la mayoría de ellas eran desconocidas hasta el momento.

Quizás al ser un proyecto no destinado a cliente final, sino que para uso interno, no resulte ser tan enriquecedor, además de realizarse de forma casi puramente individual y no en un equipo de desarrollo.

A excepción del diseño Web (maquetación y estilos de la Web), y la ejecución de algunas de las pruebas (usabilidad), además del uso de diversos recursos obtenidos vía Web, todo ha sido trabajo del autor de forma individual.

Problemas encontrados

Los principales problemas encontrados han sido las esperas al trabajo de otras personas en el proyecto, al ser de uso interno no se le da la misma importancia que a aquellos que van destinados a cliente final.

En el calendario podemos ver como afectan estas esperas a la realización del proyecto, a continuación detallamos cada uno de los problemas encontrados.

Esperas al diseño Web

La empresa sólo dispone de un diseñador Web, por lo que el diseño Web se nos facilitó con retraso, en el periodo posterior a las pruebas, cuando el sistema estaba *completamente* funcional.
Hubo que paralizar la documentación para la acomodación del sistema al nuevo diseño Web, lo cual nos llevo tres días de retraso.

Esperas a la implementación del servicio Web

Los servicios Web que se suponían implementados, no lo estaban (al menos uno), por lo que la instalación del sistema se retrasó bastante más de lo esperado.

Cambio de herramientas

Tras haber realizado una formación previa en el estudio de las herramientas a utilizar (en concreto **Struts**), y estando a unos días de empezar con la implementación, se modificó la elección de ésta herramienta, dado que un principio se pensó en que sería la que la empresa utilizaría en un futuro, siendo la mejor opción para formarnos.

Sin embargo la tecnología que se sugirió posteriormente fue **Spring**, que fue nuestra primera elección por ser más nueva y basada en Struts (Corrigiendo fallos de diseño), por lo que no hubo ningún impedimento para cambiar, salvo el consecuente retraso de un par de días.

Planificación no tenida en cuenta

Formación

La formación no se ha tenido en cuenta en la planificación previa, así como el estudio de las diversas alternativas y tecnologías, esta fase de formación se solapó levemente con la implementación, tratando de realizar los ejemplos simples de los diversos tutoriales utilizados transformándolos levemente para que fueran útiles en cierta medida para nuestro sistema final.

Tampoco se tuvo en cuenta la formación en el entorno de trabajo, para realizar diversas tareas con las herramientas utilizadas por la empresa.

Instalación del entorno de desarrollo

No se tuvo en cuenta tampoco la puesta a punto del entorno de desarrollo e instalación del software necesario, entre ellos la instalación de una base de datos local, así como la puesta a punto de las herramientas y tecnologías utilizadas.

Salidas a la universidad

No se han tenido en cuenta en la planificación las diversas salidas a la universidad para las visitas al tutor y entrega de documentos, al estar en la empresa en jornada completa y partida, el asistir a la universidad nos quita horas de trabajo de la planificación.

Trabajo futuro

Pese a que los objetivos del proyecto han sido cumplimentados correctamente, siempre es posible mejorar y ampliar lo desarrollado, exponemos a continuación diversos puntos en los que puede mejorarse el sistema

Mejorar la usabilidad del sistema

Ampliar las formas de acceso y selección de datos, de usabilidad a la hora de manejar grandes cantidades de datos.

Ampliar funcionalidad

Ampliar la funcionalidad del sistema siempre es posible, aunque no siempre recomendable dado que puede resultar una aplicación con demasiada funcionalidad sin uso. Quizás una gestión más sofisticada podría ser útil, organizando los datos que se visualizan en forma de árbol de directorios, dejando en manos del usuario la capacidad de ordenarlos. Por ejemplo ordenar al personal por áreas, o los proyectos en terminados/no terminados, ó las auditorías por Estado en la visualización del informe, etc.

También podría ampliarse la diversidad de informes y métricas que el sistema ofrece, con más estadísticas, etc.

Mejorar el rendimiento

También es una tarea fundamental aunque no requerida por el sistema, dado que es de uso interno (y no va a ser usado por muchas personas en principio) la mejora del rendimiento del sistema.

Calendario final

Hemos desarrollado un calendario real en unas hojas Excel para poder actualizarlo con facilidad día a día, una aproximación resumida del calendario en un diagrama grantt sería el siguiente.

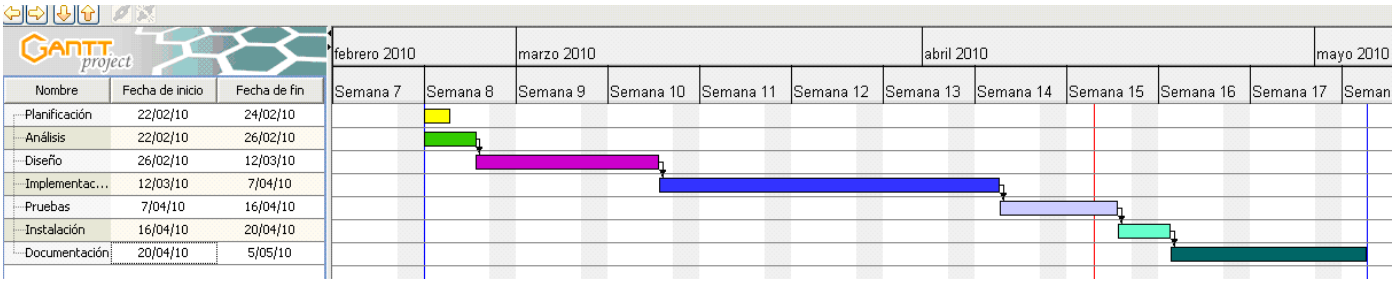


Ilustración 8 – Planificación inicial

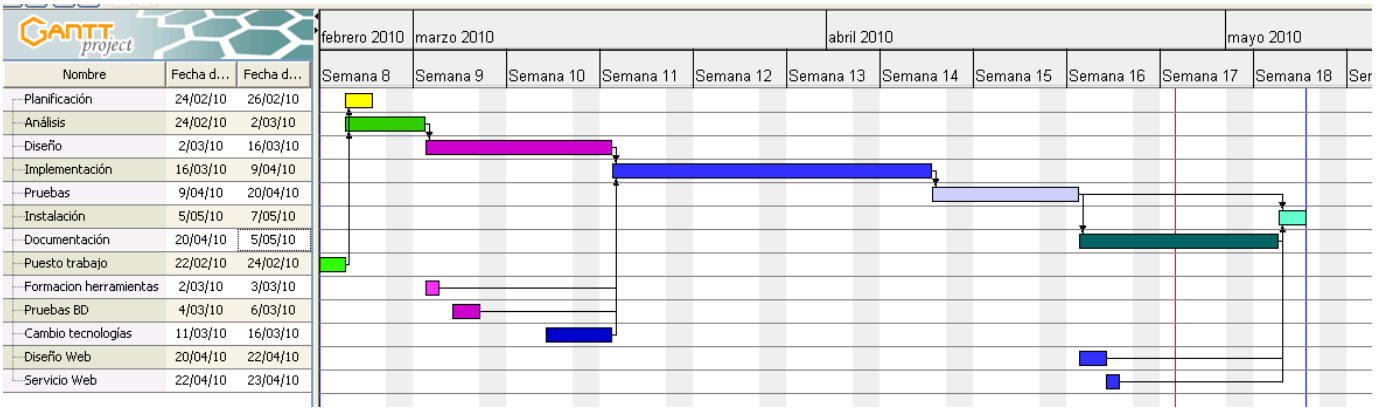


Ilustración 9 - Re-planificación

Arriba observamos la planificación inicial, y abajo la re-planificación debido a las tareas pequeñas que corresponden a tareas no planificadas que interfieren con la planificación.

Las más destacadas son las dos últimas incidencias, correspondientes a las esperas de trabajo ajeno, que tienen que ver con la implementación y tienen lugar en el apartado posterior a las pruebas e instalación del sistema, por lo que retrasa completamente la instalación del software y la ejecución de algunas pruebas.

La re-planificación consiste en redactar la documentación, justo después de las pruebas y retrasar la instalación del sistema a la semana 18.

Podemos observar también que el inicio del análisis y planificación se ve retrasado por la formación en el puesto de trabajo.

Observamos también que el cambio de tecnologías y la formación interfiere directamente con el diseño (solapándose), dejando menos tiempo para el diseño. Lo

mismo ocurre con la implementación de aquellas partes del sistema afectados por las esperas, que se solapa con la documentación.

En la siguiente figura podemos comparar en horas la planificación real de lo realmente realizado, podemos observar que la formación se ha llevado gran parte del periodo planificado para el análisis, diseño y planificación, mientras que la implementación ha requerido más tiempo del esperado debido a las esperas. Las pruebas llevarón más tiempo del esperado, dado que no se contempló la ejecución de planes de prueba genéricos.

El número de horas finales del proyecto han sido unas 460, de las cuales 40 son deformación y estudio de tecnologías, no tenidas en cuenta en la planificación inicial, por lo que no nos hemos desviado demasiado.

En la siguiente imagen tenemos una comparación en horas de lo planificado con lo realmente realizado.

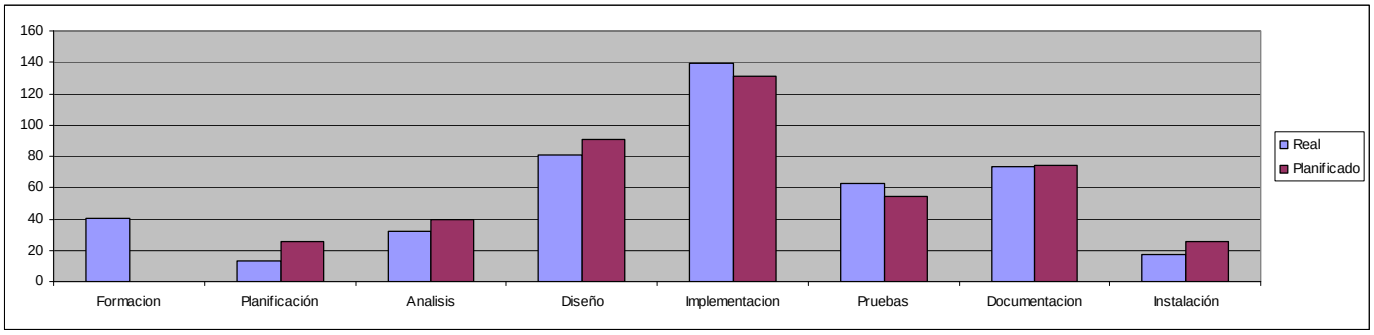


Ilustración 10 – Horas planificadas / reales

Valoración personal

El proyecto realizado ha sido muy enriquecedor, bajo nuestro punto de vista el trabajo realizado cumple perfectamente las expectativas y objetivos del proyecto marcado.

Las estimaciones realizadas pese a ser las primeras que se realizan de forma más profesional y seria han sido bastante acertadas.

Pese a no tener formación previa por parte de la empresa (toda la formación realizada ha sido auto-aprendizaje), el progreso realizado durante el desarrollo del proyecto ha sido en general bastante bueno, salvo en casos puntuales donde ha costado algo más de lo esperado aprender a utilizar alguna de las herramientas ó solventar algunos de los problemas encontrados.

Bibliografía

Documentación Java

http://www.sun.com/java/

Documentación JavaScript

http://www.javascript.com/

Documentación Ajax

http://www.w3schools.com/Ajax/Default.Asp

Documentación JSP

http://java.sun.com/products/jsp/

Documentación CSS

http://www.w3c.es/divulgacion/guiasbreves/HojasEstilo

Tutorial de una aplicación Web con Spring

http://static.springsource.org/docs/Spring-MVC-step-by-step/part1.html

Tutorial de Spring Security

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=utilizaciondegruposenspringsecurity

Tutorial Timer Task

http://www.javapractices.com/topic/TopicAction.do?Id=54

Tutorial Servlets

http://www.aprendergratis.com/aprenda-servlets-de-java-como-si-estuviera-en-primero.html

Índice de figuras y tablas

MEMORIA

Referencias

[1] http://tomcat.apache.org/ - Apache tomcat

[2] https://opensvn.csie.org/ - OpenSVN, Subversion

[3] http://tortoisesvn.tigris.org/ - Tortoise

[4] http://www.microsoft.com/spain/windows/internet-explorer/ - Internet Explorer

[5] http://www.mozilla-europe.org/es/firefox/ - Mozilla Firefox

[6] www.eclipse.org – Eclipse IDE

[7] http://www.pacestar.com/edge/index.html - Edge Diagrammer

[8] http://www.ganttproject.biz/ - Grantt Project

[9] http://testlink.org/ - Testlink

[10] http://java.sun.com/javaee/technologies/persistence.jsp - JPA

[11] http://www.hibernate.org/ - Hibernate

[12] http://ws.apache.org/wss4j/ - Web Service Security for Java

[13] http://ws.apache.org/axis/ - Apache Axis Web Services

[14] http://www.springsource.org/ - Spring Framework

[15] http://static.springsource.org/spring-security/site/index.html - Spring Security

[16] http://www3.uji.es/~mmarques/f47/apun/node87.html - Diseño lógico

[17] http://java.sun.com/blueprints/patterns/MVC.html - Patrón de diseño MVC (Modelo Vista Controlador)

[18] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

[19] http://static.springsource.org/spring/docs/2.0.x/api/org/springframework/web/servlet/mvc/multiaction/MultiActionController.html - MultiActionController

[20] http://www.w3.org/TR/wsdl - Web service description language (WSDL)

[21] http://webservices20.blogspot.com/2008/10/wcf-wse-interoperability.html - Problemas WSE 3.0

[22] http://ajaxpatterns.org/Suggestion - Ajax Suggestion

[23] http://java.sun.com/j2se/1.4.2/docs/api/java/util/TimerTask.html - TimerTask

[24] http://www.junit.org/ - JUnit Test