Grado en Ingeniería Informática

Trabajo Final de Grado

Implementación de un sistema automático de despliegue de actualizaciones

Autor:
Aarón Nadal Bosch

Supervisor:
Héctor Sánchez Sanmartín

Tutor académico:
Reyes Grangel Seguer

Fecha de lectura: 21 de julio de 2014
Curso académico 2013/2014

Resumen

Durante el proyecto se han realizado el análisis, diseño, desarrollo, pruebas y despliegue de un sistema automático de actualizaciones para ARO, el ERP propietario de Planatec Software S.L. El resultado, una aplicación web implementada utilizando PHP y Symfony2, sustituye al antiguo sistema y ofrece mejoras funcionales.

El desarrollo se centra en el módulo principal, que actúa como repositorio de versiones, aunque incluye la integración con dos módulos externos que se comunican y realizan acciones sobre el mismo. El primero de ellos se encarga de publicar nuevas actualizaciones y el segundo de desplegarlas en los clientes.

Palabras clave

Actualización, Versiones, Sistema automático, Aplicación web, Symfony2

Keywords

Update, Versions, Automatic system, Web application, Symfony2

Índice general

Índice de figuras

Capítulo 1

Introducción

Índice

1.1. Contexto y motivación

Es bien conocido que para que una empresa ofrezca un servicio rápido y eficaz, la automatización de procesos es un factor indispensable. Asimismo, para una empresa como Planatec Software S.L. dedicada principalmente al desarrollo y mantenimiento de ARO, un sistema ERP propio enfocado al sector del azulejo, es vital mantener actualizado su producto en función de las nuevas necesidades de los clientes así como de los cambios en el entorno económico y legal.

A medida que la empresa crece, el número de clientes aumenta. Tareas que antes debían realizarse unas pocas veces por actualización, ahora prácticamente superan el medio centenar.

Con cada actualización se deben descargar los ficheros necesarios en los servidores de cada cliente, ejecutar una serie de rutinas en la base de datos de cada máquina, instalar la versión de ARO actualizada, cambiar algunos ficheros de configuración en caso de ser necesario, desplegar el conjunto de nuevos informes, y un largo etcétera de otros *'cambios insignificantes'*.

Con varias actualizaciones del ERP por semana, es impensable realizar todas las tareas vinculadas a cada actualización manualmente para cada uno de los casi cincuenta clientes que posee la empresa. Lo que para un solo cliente son *'cambios insignificantes'* para medio centenar se convierten en tareas tediosas y repetitivas que se llevan consigo gran parte del tiempo de los desarrolladores de la empresa.

A causa de esto, Planatec Software S.L. cuenta, a día de hoy, con un sistema de actualizaciones que automatiza gran parte de este trabajo.

Sin embargo, el continuo crecimiento de la empresa ha causado que este sistema quede obsoleto. Las facilidades que ofrece ya no son suficientes para el equipo de desarrollo ni para el jefe del proyecto. Es verdad que este sistema automatiza muchas tareas, pero aun quedan otras que son muy tediosas de realizar manualmente. Además, la tecnología sobre la que está desarrollado es bastante antigua y poco flexible: implementar las nuevas necesidades que han ido surgiendo a los desarrolladores a lo largo del tiempo o bien no es posible o bien supondría un sobrecoste demasiado elevado.

Debido a esto, la empresa se ha planteado realizar un proyecto que renueve completamente el sistema de actualizaciones actual, haciéndolo migrar a una plataforma tecnológica más moderna, flexible y mantenible.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar una aplicación web interna para la empresa que:

⇒ Renueve el núcleo del sistema de actualizaciones de ARO, el ERP de la empresa.

⇒ Mantenga la compatibilidad con los sistemas que interactúan con el sistema actual.

⇒ Automatice una serie de procesos que actualmente se llevan a cabo de forma manual.

⇒ Ofrezca un mantenimiento más sencillo y una mayor flexibilidad que el sistema actual.

⇒ Añada funcionalidades que faciliten las tareas de los empleados.

Siguiendo la metodología de trabajo de la empresa, el proyecto se va a dividir en cuatro paquetes de trabajo, partiendo de los objetivos que se desean obtener en cada uno de ellos:

1. *Análisis de requisitos.* Determinar las necesidades de la empresa y obtener un documento con los requisitos que debe cumplir el producto.
2. *Diseño e implementación.* Realizar un diseño de la aplicación y obtener un producto funcional.
3. *Pruebas.* Lanzar pruebas sobre el producto para determinar que funciona como se espera y obtener una aplicación robusta y estable.
4. *Despliegue.* Poner en marcha el producto desarrollado y probado para su utilización en el entorno de explotación.

1.3. Estructura de la memoria

Esta memoria se divide en siete grandes capítulos, sin contar el capítulo de introducción: *Descripción del proyecto*, *Planificación*, *Análisis de requisitos*, *Diseño e implementación del sistema*, *Pruebas y despliegue*, *Conclusiones y trabajo futuro* y *Anexos*.

En el primero de ellos, *Descripción del proyecto*, se explica con detalle el proyecto. Se analiza la situación inicial, se presentan las mejoras que se deben realizar en la implementación del nuevo sistema y se enumeran las distintas tecnologías que se van a utilizar durante el desarrollo.

En el capítulo de *Planificación* se presenta la planificación del proyecto, tanto temporal como de recursos. Además, se describe de una forma detallada cuál va a ser la metodología de trabajo que se va a seguir y el seguimiento realizado durante el proyecto.

El tercer capítulo es el de *Análisis de requisitos*. En él se expone el análisis realizado en la primera fase del proyecto, así como los resultados obtenidos: diagrama de casos de uso, diagrama de clases y prototipos de la interfaz.

El siguiente capítulo, *Diseño e implementación*, describe cuál ha sido el flujo de trabajo seguido durante el desarrollo del proyecto. En ella se detalla cada una de las actividades realizadas.

El capítulo de *Pruebas y despliegue* engloba tanto las validaciones realizadas para comprobar el correcto funcionamiento del sistema como la posterior puesta en marcha de la aplicación en el entorno de explotación.

En el sexto capítulo, *Conclusiones y trabajo futuro*, se exponen las principales conclusiones obtenidas tras el desarrollo del proyecto, así cómo las lecciones aprendidas y los posibles proyectos futuros.

Por último, al final de la memoria, se adjuntan una serie de anexos con información adicional y con los documentos obtenidos durante el desarrollo del proyecto.

Capítulo 2

Descripción del proyecto

Índice

A lo largo de este capítulo se realiza una explicación del contexto del proyecto. En ella se presenta la empresa, se evalúa el producto existente, su estructura, sus puntos fuertes y débiles, y se describe la lista de cambios y mejoras propuesta para la nueva versión del sistema de actualizaciones.

2.1. La empresa, Planatec Software S.L.

Planatec Software S.L. es una empresa que tiene como objetivo principal ofrecer soluciones software a pequeñas y medianas empresas.

Se trata de una pequeña organización con no más de diez empleados que dividen sus esfuerzos en cuatro áreas bien diferenciadas:

⇒ *ERP.* La mayoría de esfuerzo de la empresa va dirigido a mantener ARO, el sistema ERP desarrollado en su totalidad por los empleados de Planatec Software S.L.

⇒ *Web.* Otro ámbito importante para la empresa es el diseño web. Planatec Software S.L. ofrece a sus clientes un servicio de implementación de páginas personalizadas.

⇒ *A medida.* Un tercer servicio que ofrece la empresa es el desarrollo de aplicaciones de escritorio a medida.

⇒ *Sistemas Hardware.* Finalmente, una pequeña parte de este esfuerzo se centra en instalar y reparar sistemas hardware a clientes particulares o empresas.

2.2. Situación inicial

Tal y como se ha mencionado en la introducción, la plantilla de Planatec Software S.L. cuenta en la actualidad con un sistema de actualizaciones que les permite publicar y desplegar nuevas versiones de ARO de manera semiautomática. Pero... ¿qué es ARO exactamente y cuál es su estructura básica?

2.2.1. ARO, el sistema ERP

ARO es el ERP propietario de Planatec Software S.L. Programado en Delphi y con Interbase como SGBD, se trata de un sistema integrado de gestión enfocado especialmente al sector del azulejo. Con casi medio centenar de clientes, el éxito del producto se debe, según el jefe del proyecto, al servicio de atención al cliente que ofrece la empresa. Atienden llamadas, solventan dudas y escuchan las propuestas y nuevas necesidades de sus clientes para incorporarlas rápidamente a su producto.

Bajo el término ARO existen tres conceptos que se pueden considerar primarios: módulos, parches e informes. Se trata de los componentes básicos que lo conforman.

Módulos

Los módulos son las piezas individuales en las que se organiza el ERP. Cada uno de ellos se especializa en un área de la empresa y todos juntos conforman el ERP completo.

Es importante que un sistema con la envergadura de ARO se organice en módulos independientes. Este hecho ofrece una serie de ventajas, tanto a los desarrolladores como a los clientes.

Al dividir el todo en varias partes, las tareas de mantenimiento e introducción de nueva funcionalidad se facilitan considerablemente. El programa a modificar no es tan grande, por lo que las zonas de código en las que se deben realizar los cambios se encuentran más acotadas y son más sencillas de localizar. Además, al tratarse de piezas independientes, los desarrolladores tienen la seguridad de que los cambios realizados en un módulo no van a afectar bajo ningún concepto a cualquiera de los demás.

Del mismo modo, dado que las necesidades de los clientes no son las mismas para todos, el hecho de contar con módulos independientes ofrece a los compradores la posibilidad de contratar únicamente los módulos que van a serles realmente útiles.

Para gestionar qué módulos se deben actualizar y cuáles no, cada uno de ellos cuenta con un sistema de versiones que identifica cuál es su última instancia funcional. Este sistema proporciona, además, facilidades para restaurar versiones anteriores en caso de algún error crítico introducido en alguna versión posterior.

Al tratarse de módulos independientes, no todos avanzan a la misma velocidad, pues hay unos módulos que necesitan más actualizaciones que otros, ya sea porque son más demandados o porque el contexto que los rodea evoluciona más rápidamente. De esta forma, es posible encontrarse con un módulo en la versión 4.21 y otro en la 4.24, por ejemplo.

Asimismo, cada versión de un módulo depende de un estado mínimo de la base de datos y de unos informes en concreto para funcionar correctamente.

Parches

Parches es el término que utilizan los desarrolladores de Planatec Software S.L. para referirse a las versiones de la base de datos de ARO. Este término se debe a que dichas versiones se componen de parches SQL que se ejecutan sobre el servidor de Interbase para actualizar su esquema.

Como se ha mencionado en el punto anterior, ARO se divide en módulos independientes. Esto significa que cada módulo puede funcionar correctamente sin la necesidad de convivir con ningún otro. Sin embargo, los datos que utilizan no son propios de cada uno. Es lógico pensar que los datos introducidos desde un módulo deben poder consultarse, e incluso modificarse, desde los demás. ¿Por qué? Porque es posible que un módulo necesite consultar, por ejemplo, todas las facturas para realizar una estadística de ventas. Si las facturas se crean desde módulos distintos en diferentes circunstancias, la estadística no estará completa si no se obtienen los datos generados por todos los módulos.

Para gestionar estos casos correctamente, ARO utiliza el concepto de datos compartidos. Gracias a ello, el ERP cuenta con un sistema de persistencia común a todos los módulos e igual en todos los clientes, independientemente de qué módulos tengan contratados.

Aunque el hecho de que todos los clientes tengan la misma estructura en el sistema de persistencia pueda parecer contraproducente, realmente no lo es. Es verdad que si un cliente tiene contratado únicamente un módulo contará con una gran cantidad de tablas innecesarias en el sistema. Sin embargo, se trata de tablas vacías que no ocupan espacio y que no causan ningún tipo de problema de rendimiento al cliente.

Mediante el método de persistencia compartida, el cliente es capaz de insertar datos desde un módulo y poder consultarlos desde cualquier otro que pueda trabajar con ellos sin ninguna gestión adicional. Además, facilita a los desarrolladores de Planatec Software S.L. las tareas de contratación de nuevos módulos, ya que no es necesario realizar ningún cambio en el sistema de persistencia cuando se instalan.

Al igual que los módulos, los parches también cuentan con un sistema de versiones. Este sistema permite identificar qué parche es el último publicado y facilita la gestión de los mismos. Es este sistema de versiones el que se utiliza para crear las relaciones entre una versión de un módulo y el parche mínimo necesario para que esta funcione correctamente.

Informes

Una de las características que ofrece ARO es la creación de informes. Gracias a ella, los usuarios pueden obtener de una manera rápida y sencilla copias impresas de estadísticas, documentos comerciales, nóminas, facturas, etcétera.

El funcionamiento de los informes es sencillo. Cuando hace falta uno nuevo, los desarrolladores de Planatec Software S.L. crean una plantilla XML y la colocan en los ordenadores de cada cliente. Esta plantilla contiene campos variables que se rellenan con los datos pertinentes cuando el cliente solicita dicho informe desde cualquier módulo. El resultado final es un documento PDF listo para imprimir.

A diferencia de los módulos y los parches, los informes no cuentan con sistema de versiones. Al ser una incorporación más reciente, todavía no se ha tenido la necesidad de implementarlo.

Pero entonces, ¿cómo se gestionan las dependencias entre un módulo y los informes mínimos necesarios? Sencillamente, se despliega el conjunto completo de informes. Mediante este método los desarrolladores de Planatec Software S.L. se aseguran de que los informes necesarios se encuentran en la máquina del cliente.

Existen dos tipos de informes:

⇒ *Informes comunes.* Son los informes que se reutilizan en todos los clientes. Suelen ser aquellos en los que no importan los detalles visuales, sino el contenido. Informes de este tipo pueden ser los de estadísticas internas.

⇒ *Informes personalizados.* Son los informes que se personalizan para cada cliente. Surgen cuando los cliente necesitan un diseño personalizado o cuando un cliente necesita un informe que no van a utilizar los demás. Informes de este tipo pueden ser las facturas, en las que cada cliente necesita un logotipo y una estructura propia.

Tanto unos como los otros son distribuidos de forma manual a todos los clientes, uno por uno. Cuando hay una nueva versión, los desarrolladores de Planatec Software S.L. se conectan a los ordenadores del cliente y despliegan los informes manualmente. Realizar este trabajo consume más de una hora de esfuerzo por cada actualización publicada.

Como se verá más adelante, una de las posibles mejoras del proyecto es dotar al nuevo sistema de actualizaciones de funcionalidad para realizar este despliegue de manera automática, en base al siguiente objetivo definido para este proyecto:

> *"Desarrollar una aplicación web interna a la empresa que automatice una serie de procesos que actualmente se llevan a cabo de forma manual."*

2.2.2. El sistema de actualizaciones de ARO

El sistema de actualizaciones de ARO se compone de tres módulos claramente diferenciados: el repositorio central, el script publicador y el script actualizador (ver Figura 2.1). A continuación se detallan las características de cada uno de ellos.

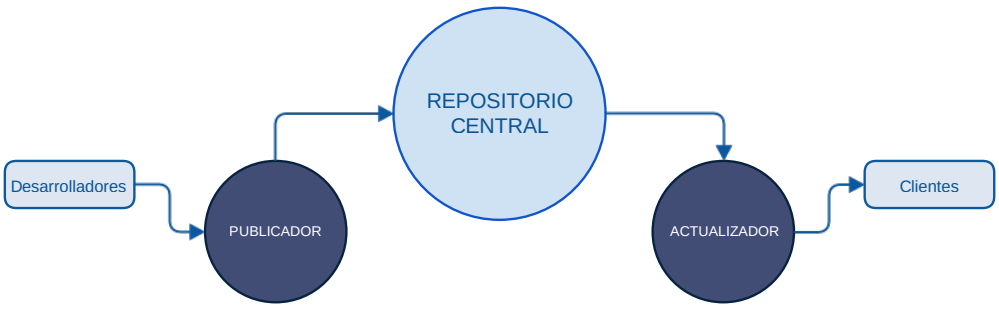

Figura 2.1: Esquema del sistema de actualizaciones.

El repositorio central

El repositorio central es el corazón del sistema de actualizaciones. Es el pilar base sobre el que interactúan todos los demás. En él se registra cada nueva versión, tanto de los módulos como de la base de datos, y se almacenan los ficheros que las componen.

Está implementado sobre Zope [9][8], una plataforma software desarrollada en Python que permite al equipo de desarrollo gestionar un sistema de ficheros y realizar operaciones sobre él. Aunque la funcionalidad que ofrece es muy rígida y depende enormemente de la estructura del sistema de ficheros, hasta hace poco cubría eficazmente las necesidades de los desarrolladores.

A medida que la empresa ha ido creciendo, la rigidez del sistema se ha ido convirtiendo en un obstáculo y la dependencia que existe entre el sistema y la estructura de la jerarquía de ficheros no ha sido más que un sólido impedimento a la hora de modificar ciertos aspectos o añadir funcionalidad.

La poca flexibilidad del sistema ha obligado a realizar ciertas tareas de forma manual al no ofrecer posibilidades de automatización. Este hecho es un lastre cada vez mayor para los empleados que las realizan, ya que el número de clientes va en aumento y junto con él, el número de veces que se deben repetir estas tareas.

Aun así, a pesar de su rigidez, el repositorio central realiza gran cantidad de tareas de forma automática y desatendida. Algunos ejemplos pueden ser los siguientes:

⇒ Registra y almacena nuevas versiones de módulos y de parches.

⇒ Ofrece la posibilidad de comprobar remotamente si una versión ya ha sido publicada o no.

⇒ Permite comprobar de manera programática si existen versiones posteriores a una dada.

⇒ Da la posibilidad de descargar las distintas versiones publicadas.

⇒ Implementa un sistema de comprobación de la integridad de los ficheros descargados.

El script publicador

El publicador es un pequeño script implementado en Python y utilizado exclusivamente por los desarrolladores de Planatec Software S.L. Con él, la tarea de publicar nuevas versiones de parches o de módulos en el repositorio central se ve simplificada.

Los desarrolladores, tras iniciar la ejecución del script, introducen el módulo para el que desean publicar una nueva versión.

A partir de este momento, todo el proceso de publicación es automático. El publicador obtiene la nueva versión del módulo indicado, recupera la versión mínima necesaria de parches y realiza la solicitud de publicación al repositorio central, proporcionando todos los datos y ficheros necesarios.

El script actualizador

El script actualizador, implementado también en Python, es algo más complejo que el anterior. Este script se distribuye a cada cliente que contrata el ERP y se ejecuta de forma desatendida cuando el sistema no está siendo utilizado. A grandes rasgos, consta de dos partes:

⇒ *Descarga.* De manera automática, cada noche se conecta al repositorio central en busca de nuevas versiones. Si se ha publicado alguna versión para alguno de los módulos contratados por el cliente, se inicia la descarga de la misma, así como de su versión mínima de parches.

⇒ *Actualización.* Justo tras finalizar la descarga y verificar la integridad de los ficheros, se aplica la actualización. Se ejecutan los parches de la última versión descargada, se copian los ficheros pertinentes en el directorio de despliegue, se cambian configuraciones y se realizan todos los cambios necesarios para que el sistema siga funcionando correctamente.

El proceso de actualización

En la Figura 2.2 se representa el funcionamiento del sistema de actualización de versiones de ARO antes del inicio proyecto.

1. En primer lugar, un desarrollador de Planatec Software S.L. compila una nueva versión de un módulo y la coloca en su repositorio Subversion (SVN) local.
2. Una vez preparada la versión, se ejecuta el publicador y se introduce el identificador del módulo que se desea actualizar.
3. El publicador obtiene automáticamente del repositorio SVN local tanto la nueva versión del módulo como la versión mínima de parches necesaria.
4. Tras esto, realiza la publicación en Zope, el repositorio central.
5. Cada noche, los actualizadores de cada cliente se conectan al repositorio y comprueban si existe alguna nueva versión de alguno de los módulos contratados por sus clientes.
6. En caso afirmativo, se descarga la nueva versión y se despliega en los ordenadores correspondientes.

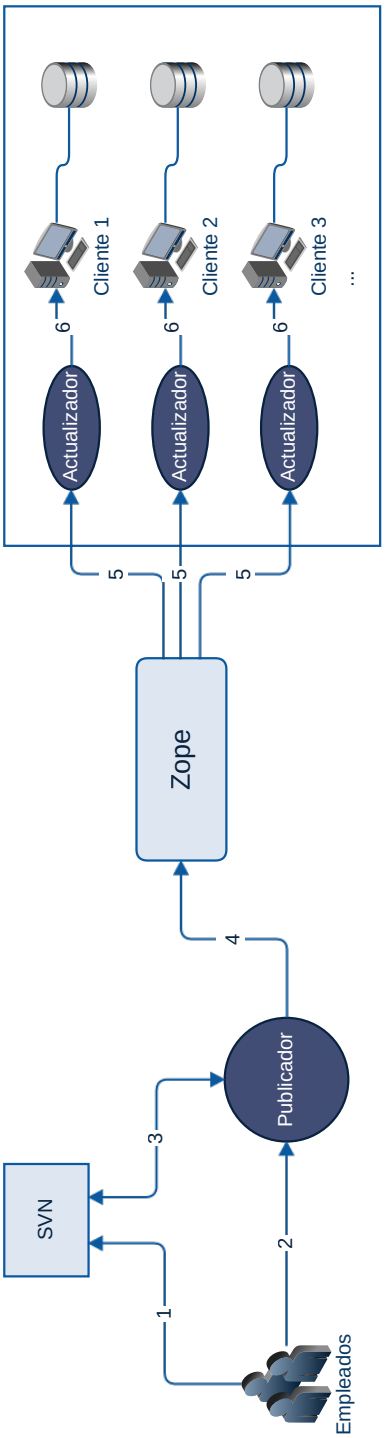

Figura 2.2: El sistema de actualizaciones inicial.

2.3. El nuevo sistema

Según los objetivos del proyecto definidos al inicio de la memoria, el nuevo sistema deberá:

⇒ **Renovar el núcleo sistema de actualizaciones de ARO.**

El nuevo sistema será una aplicación web independiente. No dependerá de ninguna plataforma externa, sino que se desplegará y mantendrá en un servidor propio de la empresa.

⇒ **Mantener la compatibilidad con los módulos dependientes.**

Para ello se deberá revisar el código de los scripts publicador y actualizador con el fin de averiguar qué tecnologías utilizan y de qué manera se comunican con el repositorio central. Dichas tecnologías y protocolos de comunicación deberán replicarse en el nuevo sistema para que ofrezca a los scripts funcionalidad similar a la que ofrece el sistema actual.

⇒ **Automatizar los procesos que se llevan a cabo manualmente.**

Algunas posibilidades de automatización son la obtención del *changelog* de una nueva versión directamente del sistema de control de versiones Subversion, la notificación por correo electrónico a los clientes de un módulo tras haber publicado una nueva versión, y la más importante de todas, el despliegue automático de informes.

⇒ **Ofrecer un mantenimiento más sencillo y mayor flexibilidad.**

Al contar con empleados experimentados y con experiencia en PHP, y al tener acceso al código fuente, se ofrecerá una mayor flexibilidad en comparación a la ofrecida por el sistema actual. Cabe destacar que, para que la aplicación sea sencilla de mantener, se requiere un buen diseño del sistema y un código fácil de entender, escalable y propenso a cambios.

⇒ **Añadir funcionalidad que facilite el trabajo a los empleados.**

Para cumplir este objetivo se dotará al sistema de nuevas funciones como la gestión de versiones desde dentro de la propia aplicación, tarea que actualmente se lleva a cabo de forma manual y muy propensa a errores; o la limpieza automática de versiones antiguas, de manera que, al sobrepasar un límite establecido para el histórico de versiones, se eliminarán automáticamente las más antiguas.

Tras la realización del proyecto, el sistema de actualizaciones deberá mantener la estructura del sistema inicial, habiendo sustituido el repositorio central Zope por el nuevo a desarrollar. El resultado esperado se muestra en la Figura 2.3.

2.4. Entorno tecnológico

En este apartado se tratan las tecnologías que se van a utilizar para el desarrollo del nuevo sistema. Entre ellas se encuentran distintas tecnologías web tales como PHP, JavaScript o AJAX, el sistema gestor de bases de datos MySQL o el lenguaje de scripting Python, así como distintos protocolos de comunicación y herramientas.

2.4.1. Tecnologías web

Debido a que el resultado de este proyecto va a ser una aplicación web, el uso de estas tecnologías es una exigencia implícita. Del mismo modo, al basarse en la arquitectura cliente-servidor, en este proyecto se van a utilizar dos tipos de tecnologías distintas:

⇒ *Tecnologías del lado del servidor:* Apache y PHP.

⇒ *Tecnologías del lado del cliente:* JavaScript, JQuery y AJAX.

Apache y PHP

Apache y PHP [3] son dos tecnologías que se usan regularmente en la empresa. Como ya se ha mencionado, parte del esfuerzo de Planatec Software S.L. va dirigido al diseño y desarrollo de páginas web. Algunas de ellas se desarrollan utilizando CMS (Content Management Systems) tales como Joomla, Drupal o Moodle, pero para desarrollar el resto se utiliza PHP.

En mi caso, el uso de estas tecnologías ha sido una exigencia. La máquina en la que se va a lanzar en primera instancia la aplicación cuenta con un servidor web Apache 2.2, por lo tanto debe ser desarrollada con un lenguaje compatible. ¿Qué mejor opción que PHP? A día de hoy es uno de los lenguajes de programación más utilizados para el desarrollo web y, además, los desarrolladores de Planatec Software S.L. ya tienen experiencia en su utilización [1][6].

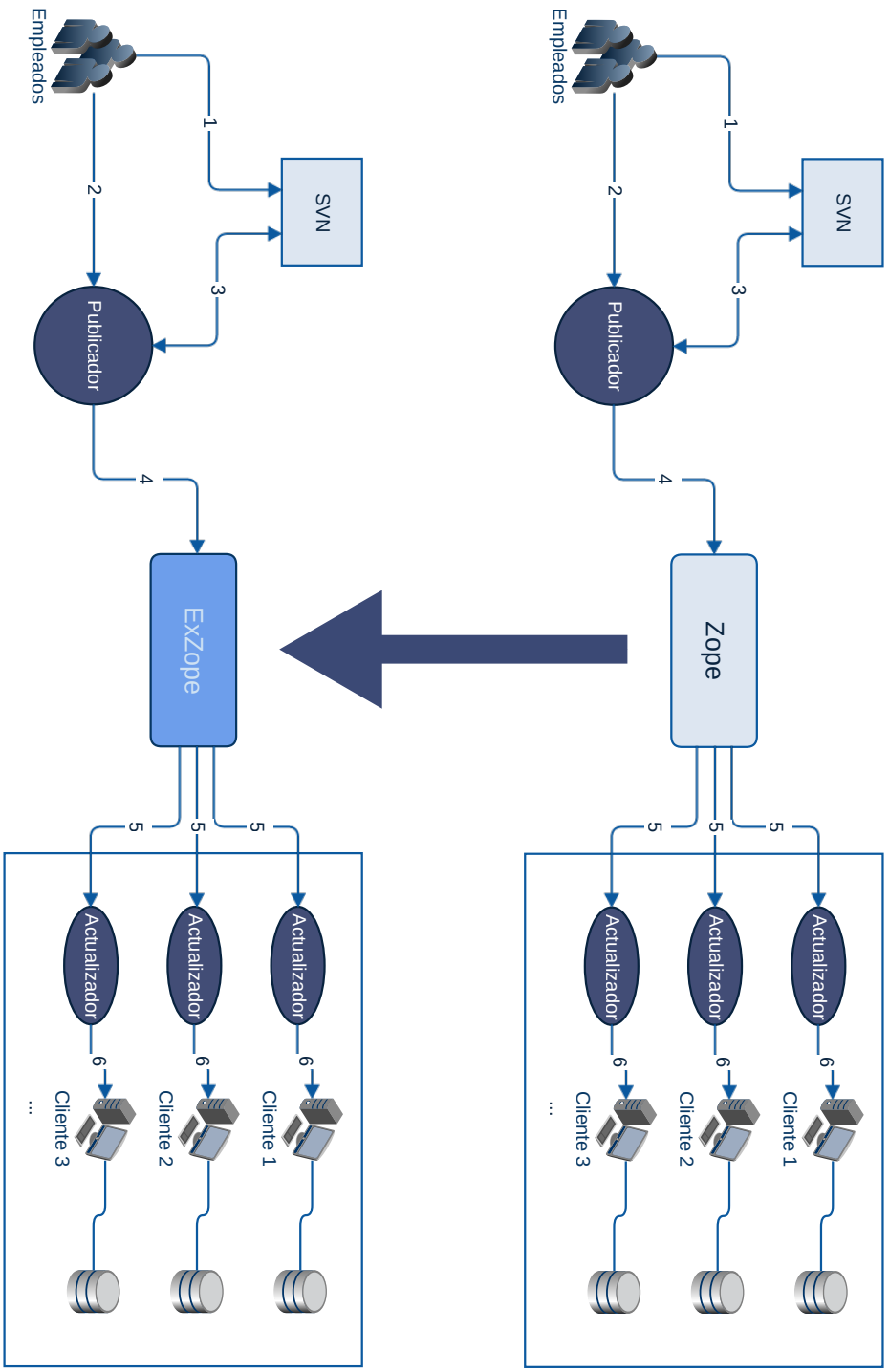

Figura 2.3: Evolución esperada del sistema de actualizaciones.

JavaScript, JQuery y AJAX

Hoy en día, una aplicación web sin contenido dinámico no es una aplicación web completa. Cada día se utilizan más tecnologías del lado del cliente para hacer más amenas y visuales las experiencias en la red. Ese es uno de los principales empeños de JavaScript y, gracias a la librería de JQuery [4], lo consigue de una manera muy sencilla.

Los scripts se ejecutan en el navegador e interactúan con los elementos HTML, modificando sus propiedades, animándolos, o desanclándolos del documento. AJAX [5], por su parte, se encarga de realizar peticiones en segundo plano al servidor para recuperar datos y proporcionárselos a los scripts.

En este apartado cabe desatacar la utilización de dos librerías muy relacionadas con estas tecnologías:

⇒ *Bootstrap* [2]. Es una librería implementada sobre jQuery y CSS que ofrece un conjunto de componentes y widgets visuales al programador.

⇒ *jQuery Validation Plugin* [11]. Es otra librería, también implementada sobre jQuery, que proporciona utilidades para realizar validaciones de formularios web en el lado del cliente.

Para la realización del proyecto, el uso de estas tecnologías no ha sido una exigencia, aunque sí que han sido recomendadas encarecidamente. De nuevo, se trata de tecnologías en auge que son utilizadas en prácticamente todos los sitios web actuales.

2.4.2. MySQL

En el contexto del desarrollo web, cuando se habla de Apache y PHP es inevitable pensar, además, en MySQL. Son tres conceptos que suelen ir de la mano [3].

Se trata de tres tecnologías que trabajan muy bien juntas. Por ejemplo, PHP cuenta con un gran número de funciones nativas para interactuar con MySQL de manera sencilla.

De nuevo, la utilización de este Sistema Gestor de Bases de Datos ha sido una obligación en el proyecto. Es el SGBD que se utiliza a nivel interno de la empresa y el que emplean todas sus aplicaciones web.

2.4.3. Symfony2

Symfony2 [7] es un *framework* basado en el patrón MVC (Modelo, Vista, Controlador). Su propósito es separar la lógica de negocio de la lógica de presentación de una aplicación web. Además, reúne un gran número de herramientas que facilitan enormemente el trabajo del desarrollador.

El *framework* integra, por ejemplo, un ORM (*Object-Relational Mapping*) llamado Doctrine que se encarga de relacionar las tablas de la base de datos con objetos PHP; o Twig, un sistema de plantillas que facilita la reutilización de código HTML.

Además de todo esto, cabe destacar que se trata de un *framework* ampliamente personalizable y con gran cantidad de librerías desarrolladas por terceros fácilmente incorporables al proyecto.

Ninguno de los empleados de Planatec Software S.L. había utilizado antes este *framework*, aunque sí que se habían informado acerca de él. Algunos de ellos habían leído su documentación, pues estaban esperando algún nuevo proyecto para introducir esta herramienta y comprobar su eficacia. Debido a esto, su uso en el desarrollo del proyecto ha sido una imposición.

2.4.4. Python

Otra tecnología muy utilizada en la empresa. La mayoría de las aplicaciones y scripts de mantenimiento están desarrollados en este lenguaje. Su sintaxis, muy limpia, permite facilitar las modificaciones de código y su rapidez y facilidad de uso hacen que el tiempo de desarrollo de una aplicación sencilla sea mucho menor.

Su uso en este proyecto viene condicionado por los scripts publicador y actualizador, desarrollados mediante este lenguaje. Estos scripts interactúan directamente con el repositorio central y va a ser necesario realizar pequeños cambios en los mismos.

2.4.5. XML-RPC

XML-RPC es un protocolo de llamadas a procedimientos remotos que utiliza el formato XML para la codificación de los mensajes y el protocolo HTTP para realizar las comunicaciones [10].

Aunque se trata de un protocolo muy simple y bastante antiguo, y aunque han surgido alternativas mejores en los últimos años, su uso viene condicionado por uno de los objetivos del proyecto:

"Desarrollar una aplicación web interna a la empresa que mantenga la compatibilidad con los sistemas que interactúan con el sistema actual."

Actualmente, tanto el actualizador como el publicador se comunican con el repositorio de versiones mediante XML-RPC, por lo tanto se debe mantener este protocolo para el correcto funcionamiento de los scripts.

2.4.6. FTP

Dado que XML-RPC no permite realizar transferencias de ficheros, solo podrá ser utilizado para comunicaciones sencillas. Con XML-RPC podremos informar al sistema de que vamos a crear una nueva versión, pero no podremos enviarle los ficheros que la componen. Para lograr este propósito debemos recurrir a un protocolo adicional que nos permita transferir ficheros entre dos ordenadores.

Por ser el protocolo más indicado para ello y por ser el que utilizan los scripts de publicación y actualización, se va a utilizar FTP para realizar las transferencias del cliente al servidor.

2.4.7. SVN

Para llevar un correcto seguimiento del código es una buena práctica utilizar algún sistema de control de versiones. En Planatec Software S.L. se utiliza Subversion desde el nacimiento de la empresa y, debido a esto, se va a utilizar de igual manera en este proyecto.

Además de para llevar un control de las versiones del proyecto, SVN también es utilizado por el sistema de actualizaciones de ARO para consultar los historiales de cambios o *changelogs* introducidos en las nuevas versiones. Para ello, se lanzan consultas al repositorio de Subversion y se solicita información acerca de los cambios introducidos en una versión concreta.

2.4.8. Trello

Trello es una herramienta web que permite al usuario gestionar un tablero Kanban personalizable. Aunque el concepto de tablero Kanban en la mayoría de ocasiones va ligado a las metodologías ágiles, es fácilmente utilizable en las metodologías tradicionales de gestión de proyectos también.

Este tipo de tableros constan, generalmente, de tres columnas: *TO DO*, *DOING* y *DONE*, en las cuales se colocan tarjetas que representan las tareas a realizar, en fase de desarrollo o ya implementadas, respectivamente.

La principal función para la que se ha utilizado esta herramienta ha sido para anotar los pequeños detalles de implementación que han ido quedando pendientes a lo largo el desarrollo del proyecto. De esta forma ha sido posible implementar hasta la más mínima especificación solicitada por el cliente sin olvidar los detalles por el camino.

En la Figura 2.4 se puede observar un ejemplo.

Figura 2.4: Captura del tablero de Trello.

Capítulo 3

Planificación

Índice

A lo largo de las secciones que conforman este capítulo se pretende mostrar la planificación del proyecto. En primer lugar, se presenta la metodología de trabajo que se ha seguido para la realización del mismo. Tras esto, se definen las tareas en las que se ha dividido el proyecto y una estimación de la duración de cada una de ellas. Además, se realiza un análisis de costes de los recursos del proyecto. Finalmente, se expone el diagrama de GANTT tanto para la planificación como para el seguimiento.

3.1. Metodología de trabajo

La metodología de trabajo que se ha seguido para el desarrollo de este proyecto se basa en el ciclo de vida clásico del desarrollo de un proyecto software, aunque ha ido evolucionando con el tiempo y se ha ido adaptando a las necesidades de la empresa. Consiste en cuatro fases claramente diferenciadas:

1. *Análisis de requisitos.*

   En esta primera fase, se estudian las necesidades del cliente para determinar qué funcionalidades debe ofrecer la aplicación y se identifican los paquetes de trabajo en los que se divide el proyecto. Estos paquetes de trabajo son conjuntos de funcionalidades relativas a un mismo concepto o con propósitos similares.

   Como resultado de esta fase se obtienen tres documentos: un diagrama de casos de uso, un diagrama de clases y un documento con los prototipos de las interfaces de usuario.

2. *Diseño e implementación.*

   En la segunda fase del desarrollo se realiza el diseño y se implementa la lógica de la aplicación. Para ello se utilizan los documentos obtenidos en la fase anterior: para el diseño de la interfaz de usuario se consultan los prototipos, para el diseño interno de la aplicación se utiliza en diagrama de clases y para asegurarse de que se implementan todos los requisitos del cliente, se contrasta el resultado con el documento de casos de uso.

   Esta suele ser la fase más extensa y se obtiene como resultado una versión completa y funcional de la aplicación.

3. *Pruebas.*

   La tercera fase consiste en diseñar una batería de pruebas y lanzarlas en primera instancia sobre un entorno de pruebas y posteriormente sobre una réplica del entorno de explotación.

   Se realizan dos tipos de pruebas: pruebas unitarias y pruebas de integración. En las primeras se prueban las unidades básicas del desarrollo, que pueden ser métodos, clases o paquetes, y en las segundas se prueba que estas unidades básicas cooperen correctamente.

4. *Despliegue.*

   Finalmente, con el producto probado y funcionando correctamente, se realiza el despliegue en el entorno de explotación.

   Además, se redacta toda la documentación necesaria, que puede variar de un proyecto a otro. Dependiendo del cliente y de la complejidad del proyecto, se pueden redactar manuales de instalación, de mantenimiento, de uso, de actualización, etc.

A pesar de que esta estructura parece estática y secuencial, la fase de diseño y desarrollo se realiza de forma iterativa: se planifica una iteración por cada uno de los paquetes de trabajo identificados en la fase de análisis.

La introducción de estas iteraciones permite al desarrollador contrastar el trabajo realizado con el cliente a medida que se va realizando la implementación. Este hecho ofrece al cliente la posibilidad de introducir pequeños cambios, tanto funcionales como relativos al diseño, sin que supongan un sobrecoste muy elevado. Asimismo, el desarrollador cuenta con la tranquilidad de que su trabajo está siendo validado por el cliente en todo momento.

3.2. Planificación temporal del proyecto

Antes de comenzar con el desarrollo del proyecto, se ha planificado el trabajo según las conclusiones obtenidas en la primera reunión con el cliente. Se han organizado las necesidades del cliente en tareas o actividades y se han estimado los costes temporales.

3.2.1. Definición de actividades

En la Figura 3.1 se muestra un diagrama EDT con el desglose de las actividades del proyecto organizadas en paquetes de trabajo.

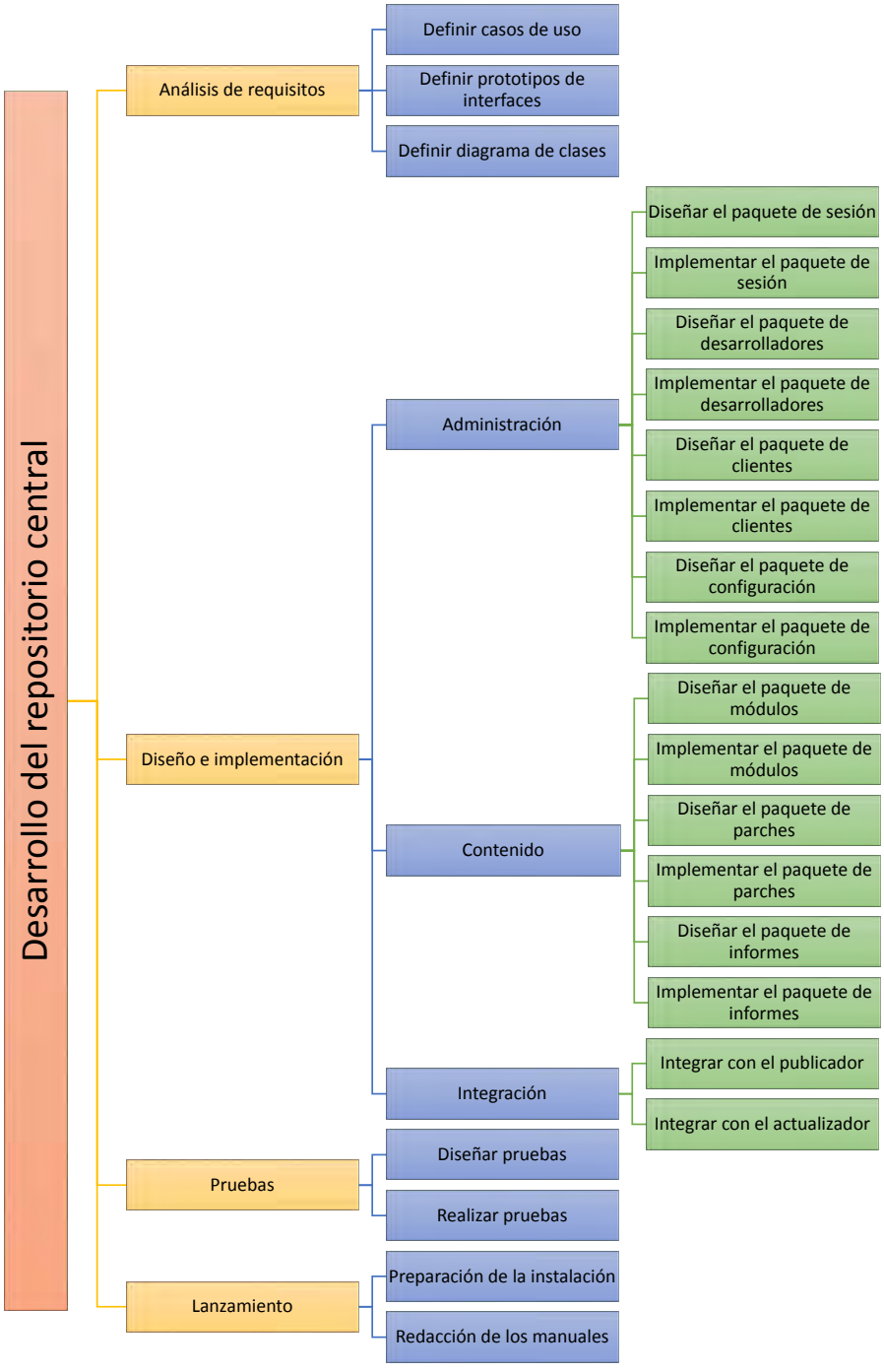

Figura 3.1: Diagrama EDT con la definición de las actividades del proyecto.

3.2.2. Estimación de costes temporales

Para la estimación de los costes temporales de cada actividad, se ha supuesto que el alumno debía realizar un horario de 5 horas al día durante 5 días a la semana. Siendo un total de 300 horas a completar durante la estancia, la duración del proyecto debería ser de 60 días (12 semanas).

Las estimaciones mostradas en esta sección se han realizado utilizando el juicio de expertos y la analogía de proyectos. Junto con el supervisor de prácticas y utilizando su experiencia en proyectos similares realizados anteriormente, se han ido determinando las duraciones aproximadas de las tareas.

La estimación del coste temporal de las actividades definidas para este proyecto ha sido la siguiente:

- Análisis de requisitos *(7 días)*.
  - ◦ Definir casos de uso *(3 días)*.
  - ◦ Definir prototipos de interfaces *(3 días)*.
  - ◦ Definir diagrama de clases *(1 día)*.
  - → Validar requisitos *(HITO)*.
- Diseño e implementación *(44 días)*.
  - Administración *(16 días)*
    - ◦ Diseñar el paquete de sesión *(1 día)*.
    - ◦ Implementar el paquete de sesión *(3 días)*.
    - ◦ Diseñar el paquete de desarrolladores *(1 día)*.
    - ◦ Implementar el paquete de desarrolladores *(3 días)*.
    - ◦ Diseñar el paquete de clientes *(1 día)*.
    - ◦ Implementar el paquete de clientes *(3 días)*.
    - ◦ Diseñar el paquete de configuración *(1 día)*.
    - ◦ Implementar el paquete de configuración *(3 días)*.
  - Contenido *(11 días)*
    - ◦ Diseñar el paquete de módulos *(1 día)*.
    - ◦ Implementar el paquete de módulos *(6 días)*.
    - ◦ Diseñar el paquete de parches *(1 día)*.
    - ◦ Implementar el paquete de parches *(1 día)*.

- Diseñar el paquete de informes *(1 día)*.
- Implementar el paquete de informes *(1 día)*.

→ Validar implementación *(HITO)*.

- Integración *(17 días)*
  - Integrar con el publicador *(9 días)*.
  - Integrar con el actualizador *(8 días)*.

→ Validar integración *(HITO)*.

- Pruebas *(6 días)*.
  - Diseñar pruebas *(2 días)*.
  - Realizar pruebas *(4 días)*.

  → Validar pruebas *(HITO)*.
- Lanzamiento *(3 días)*.
  - Preparación de instalación *(2 días)*.
  - Redacción de los manuales *(1 día)*.

  → Validar proceso de instalación *(HITO)*.

3.2.3. Diagrama de GANTT

El diagrama de GANTT resultante de la fase de planificación temporal de tareas se muestra en la Figura 3.2.

3.2.4. Estimación del coste de los recursos del proyecto

Tal y como se puede observar en el Figura 3.3, los recursos necesarios para este proyecto se pueden dividir en dos tipos:

Nombre del recurso	Tipo	Tasa estándar	Costo
Programador	Trabajo	20,00 €/hora	4.500,00 €
Diseñador	Trabajo	30,00 €/hora	1.050,00 €
Analista	Trabajo	45,00 €/hora	1.125,00 €
Licencia de Aptana Studio 3	Costo		0,00 €
Licencia gratuita de Go Mocking Bird	Costo		0,00 €
Licencia de ArgoUML	Costo		0,00 €

Figura 3.3: Listado de recursos del proyecto y costes.

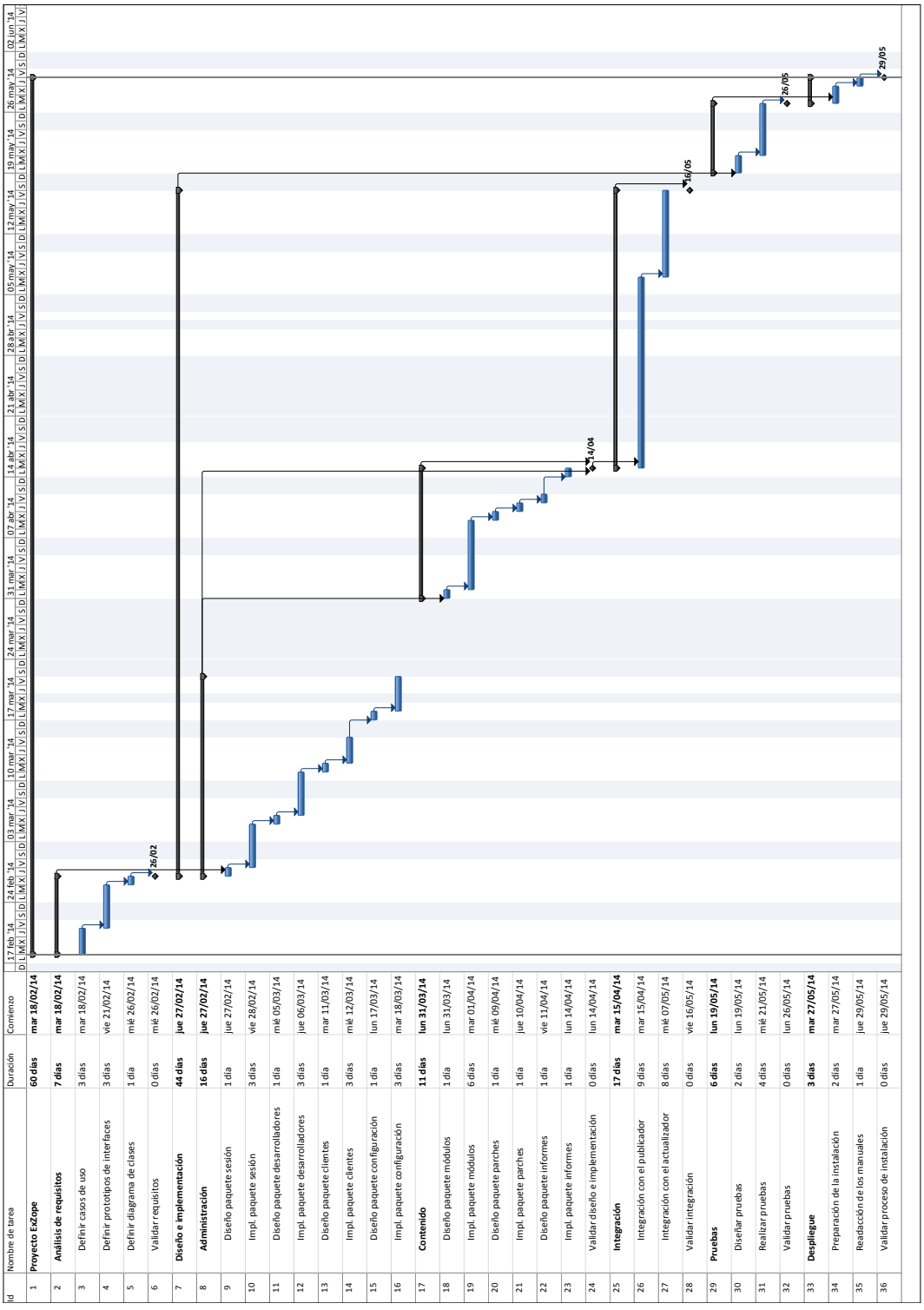

Figura 3.2: Diagrama de GANTT de la planificación temporal del proyecto.

⇒ *Licencias*. Tal y como su nombre indica, se trata de las licencias del software necesario para la realización del proyecto. En este proyecto se ha intentado en todo momento utilizar licencias gratuitas de software libre.

⇒ *Recursos humanos*. Hace referencia a las personas que han intervenido en el desarrollo del proyecto. A pesar de que en el Figura 3.3 se observan tres recursos diferentes de este tipo, ha sido el mismo alumno el que ha tomado un rol u otro en cada momento, dependiendo de la tarea a realizar. Esta división se ha efectuado con el fin de obtener una estimación de costes más cercana a la realidad.

A continuación, en el Figura 3.4, se desglosan las tareas para las que ha sido utilizado cada uno de los tres roles y el coste que han supuesto.

Nombre del recurso	Trabajo	Costo	Tasa estándar
Programador	**225 horas**	**4.500,00 €**	**20,00 €/hora**
Impl. paquete sesión	*30 horas*	*600,00 €*	
Impl. paquete desarrolladores	*15 horas*	*300,00 €*	
Impl. paquete clientes	*10 horas*	*200,00 €*	
Impl. paquete configuración	*10 horas*	*200,00 €*	
Impl. paquete módulos	*40 horas*	*800,00 €*	
Impl. paquete parches	*5 horas*	*100,00 €*	
Impl. paquete informes	*5 horas*	*100,00 €*	
Integración con el publicador	*50 horas*	*1.000,00 €*	
Integración con el actualizador	*20 horas*	*400,00 €*	
Validar integración	*0 horas*	*0,00 €*	
Diseñar pruebas	*10 horas*	*200,00 €*	
Realizar pruebas	*10 horas*	*200,00 €*	
Validar pruebas	*0 horas*	*0,00 €*	
Preparación de la instalación	*15 horas*	*300,00 €*	
Readacción de los manuales	*5 horas*	*100,00 €*	
Validar proceso de instalación	*0 horas*	*0,00 €*	
Diseñador	**35 horas**	**1.050,00 €**	**30,00 €/hora**
Diseño paquete sesión	*5 horas*	*150,00 €*	
Diseño paquete desarrolladores	*5 horas*	*150,00 €*	
Diseño paquete clientes	*5 horas*	*150,00 €*	
Diseño paquete configuración	*5 horas*	*150,00 €*	
Diseño paquete módulos	*5 horas*	*150,00 €*	
Diseño paquete parches	*5 horas*	*150,00 €*	
Diseño paquete informes	*5 horas*	*150,00 €*	
Validar diseño e implementación	*0 horas*	*0,00 €*	
Analista	**25 horas**	**1.125,00 €**	**45,00 €/hora**
Definir casos de uso	*10 horas*	*450,00 €*	
Definir prototipos de interfaces	*10 horas*	*450,00 €*	
Definir diagrama de clases	*5 horas*	*225,00 €*	
Validar requisitos	*0 horas*	*0,00 €*	

Figura 3.4: Asignación de actividades a los recursos humanos.

Teniendo en cuenta que no se ha generado coste alguno por la adquisición de las licencias software y que todo el hardware necesario ya estaba disponible antes de comenzar el proyecto, el coste total se reduce a los 6.675 euros necesarios para cubrir los recursos humanos.

3.3. Seguimiento del proyecto

A lo largo del desarrollo del proyecto se ha realizado un seguimiento del mismo para comprobar si se estaba siguiendo la planificación correctamente.

En la Figura 3.5 se puede observar el diagrama de seguimiento cuando el proyecto se encontraba al 52 % del desarrollo. En él se aprecia que la fase de análisis de requisitos ha finalizado dos días antes de lo planificado. Sin embargo, durante la fase de diseño e implementación se ha producido un pequeño retraso. Las tareas de implementación del paquete de sesión e implementación del paquete de módulos han costado más de lo planificado en primera instancia y han causado dos días de demora.

Estos retrasos se deben a que en las primeras fases del desarrollo no se han tenido los conocimientos necesarios de todas las tecnologías utilizadas, por lo que se ha tardado más tiempo en realizar algunas tareas sencillas. Del mismo modo, al realizar la planificación se ha tenido una idea general de las tareas de implementación, pero no se han conocido todos los detalles, por lo que no se ha podido planificar con total exactitud.

En la Figura 3.6 se muestra el diagrama de seguimiento una vez finalizado el proyecto. En él se puede observar que, aunque cuando el proyecto se encontraba al 50 % llevaba dos días de retraso, el desarrollo completo del proyecto se ha alcanzado tres días antes de lo esperado.

Esta ganancia de cinco días en la segunda mitad del desarrollo se ha debido, en gran medida, a la fase de diseño e implementación. En esta fase, a pesar de que la integración con el publicador ha durado un día más de lo previsto, la integración con el actualizador ha sido realizada en la mitad del tiempo estimado.

Las variaciones en la planificación de algunas tareas, como la de integración con el actualizador, se han debido a la necesidad de introducirse en un código de terceros para realizar los cambios. A priori, no es posible determinar qué dificultad supone esta inmersión en un código ajeno, ya que se desconoce qué diseño tiene el código o si es fácilmente modificable.

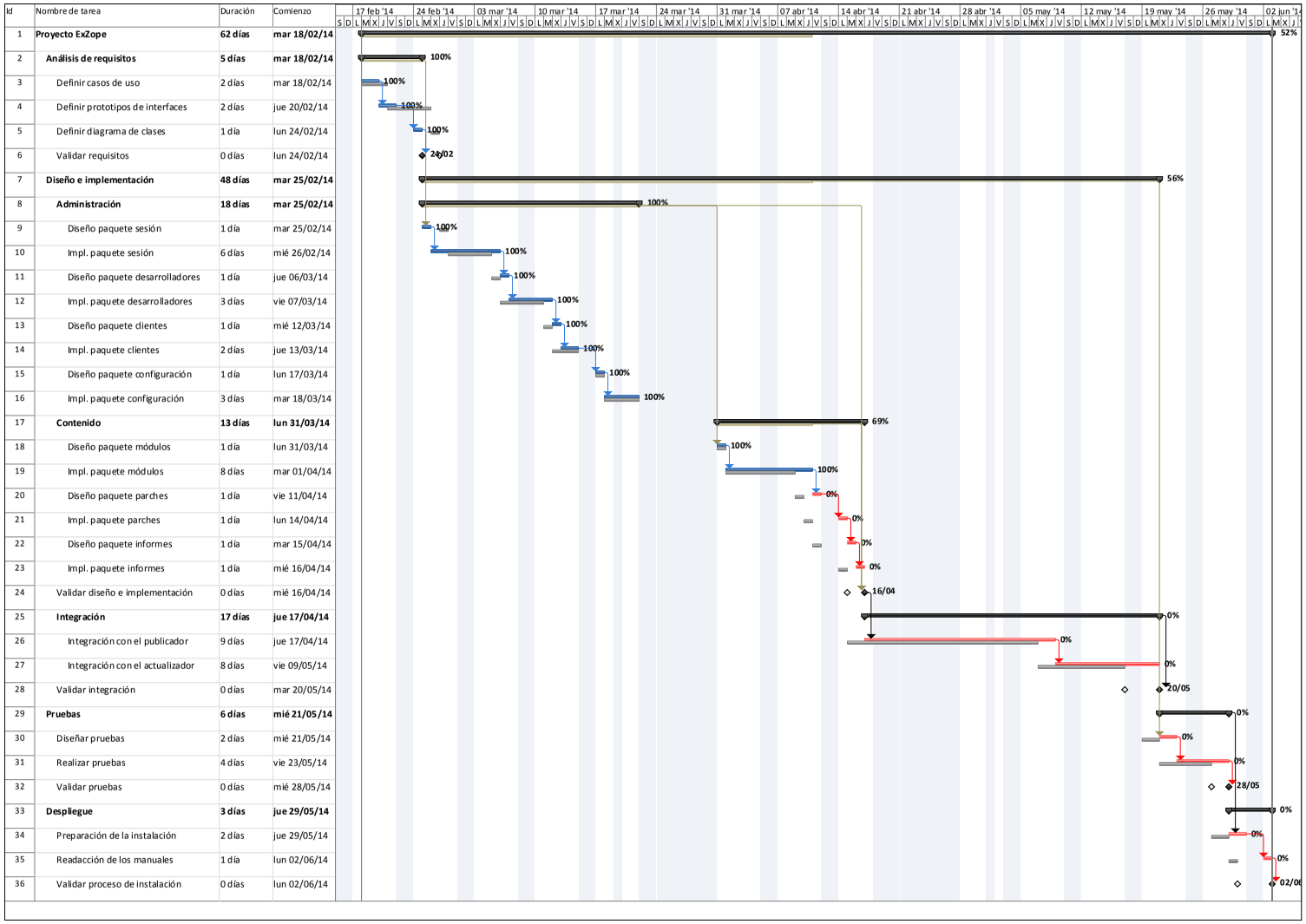

Figura 3.5: Diagrama de GANTT de seguimiento. Mitad del proyecto.